

Java图形界面开发学习笔记——从新手到高手的跨越



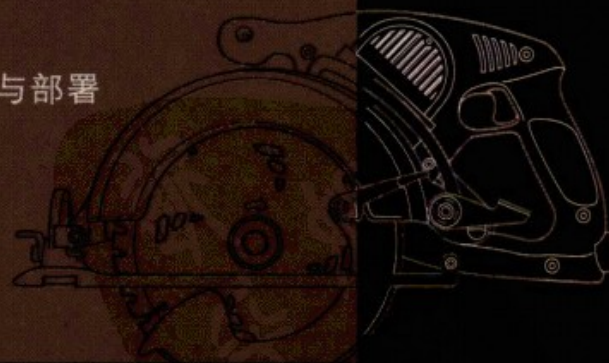
Java Swing

图形界面

开发与案例详解

王鹏 何昀峰 编著

- 涵盖Java Swing图形界面开发必须掌握的所有常用知识
- 通过**105个**具有典型性和实用价值的实例学习组件应用、开发与部署
- 通过**77个**习题巩固所学知识
- 通过1个综合实例进一步提高读者界面设计开发的能力

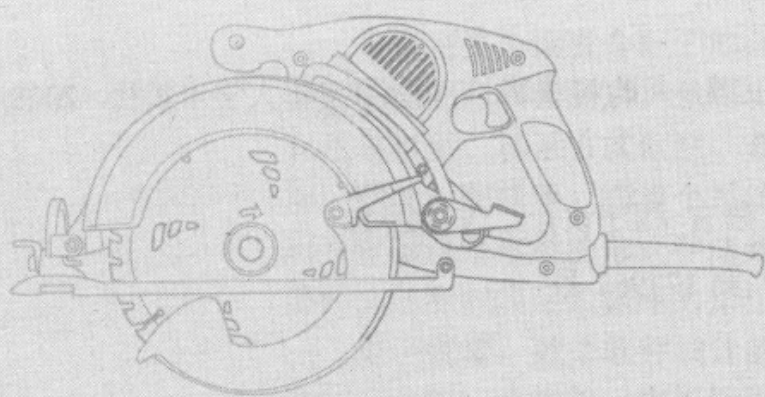


清华大学出版社

Java Swing

图形界面开发与案例详解

王鹏 何昀峰 编著



清华大学出版社
北 京

内 容 简 介

Java Swing 是目前图形界面设计的主流开发工具,本书从实用的角度出发,通过大量实例全面介绍 Java Swing 中各种组件的应用及图形界面的开发技术。

全书共 20 章,其中第 1~2 章主要介绍有关 Swing 的基础知识,包括 Swing 的基本概述、如何使用 IDE 开发 Swing 程序;第 3~15 章结合众多实例和问题介绍最常用、最有用的组件及其应用方法,包括标签和按钮组件、布局管理器组件、面板组件、列表框组件、下拉列表框组件、进度条组件、时间组件、滑块组件、分隔条组件、选取器组件、文本组件、窗口组件、对话框组件、JApplet 组件、菜单组件、工具条组件、表格组件、树组件以及组件的事件处理机制等;第 16~19 章主要讲述有关 Swing 图形开发的其他知识,包括观感器、Swing 线程与并发、模型与架构等。第 20 章通过一个综合实例使读者能够在实际开发中理解和巩固所学知识,从而提高综合应用能力。

本书几乎涵盖了目前 Java Swing 图形开发必备的所有常用知识,拥有丰富的实例,且这些实例均来自于工程实践,本书适合 Java 程序语言的初学者阅读,也可供具有一定编程经验的初级编程人员参考。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

Java Swing 图形界面开发与案例详解/王鹏,何昀峰编著.—北京:清华大学出版社,2008.12
ISBN 978-7-302-18904-6

I. J… II. ①王… ②何… III. JAVA 语言-程序设计 IV. TP312

中国版本图书馆 CIP 数据核字(2008)第 177962 号

责任编辑:夏非彼 张楠

装帧设计:图格新知

责任校对:贾淑媛

责任印制:李红英

出版发行:清华大学出版社

地址:北京清华大学学研大厦 A 座

<http://www.tup.com.cn>

邮编:100084

社总机:010-62770175

邮购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印刷者:北京密云胶印厂

装订者:三河市新茂装订有限公司

经销:全国新华书店

开本:190×260 印张:26.25 字数:672 千字

版次:2008 年 12 月第 1 版 印次:2008 年 12 月第 1 次印刷

印数:1~4000

定价:49.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。联系电话:(010)62770177 转 3103 产品编号:030557-01

前言

Preface

Java 技术的强大生命力来自于它所蕴含的面向对象和设计模式思想，由此开发出来的程序，不仅具有非常美观的艺术性，而且具有非常高的重用性，这种重用性使开发工作变得非常轻松。

Java 的 Swing 技术，因 Java 强大的生命力而被赋予非常广阔的展示舞台，Swing 技术从编码时起就能承载诸多 Java 设计理念，所以，您将在相对比较“繁琐”的 Swing 代码里体会 Java 的乐趣。“Swing 组件”、“事件处理监听器”……，这些词语目前或许仅仅是一个个饶舌的词汇，但当您阅读完本书后将会发现，它们从实现角度来看是小巧的艺术品，从应用角度来看，更能让您在使用后得到无限启发。

为了使您尽快能够编写 Swing 代码、快速掌握 Java Swing 图形开发的技巧及应用方法，在本书中笔者根据自己的实践经验，从零基础讲起，回避一些高深但不实用的知识点，专门讲解实用性强的技术。“实例化讲解”是本书的一大特色，对于书中讲述的绝大多数知识点，都会给出相应难度的实例程序，这样做的目的就是让您能够理论联系实际，所以在阅读本书时，可以采用“阅读基础知识→运行代码→深入学习”的方式，从而在实际操作中熟悉 Java Swing 图形开发技术。

本书特点

与其他书籍相比，本书具有以下特点。

- 分解知识点，逐个掌握：本书几乎覆盖 Java Swing 图形开发所需掌握的所有常用知识点。
- 实例丰富，易学易用：本书的一大特色就是拥有大量的实例，能够让读者根据实例来进一步清楚地理解所介绍的知识点。
- 图例丰富，学习轻松：在具体讲述知识点时，为了让读者从开始就能理解这个知识点的原理，本书提供了详细的图例，说明这些知识是如何设计、如何实现、如何应用的，并在图例中稍显复杂的地方，提供了详细的标注，让读者一看就明白整个知识点的设计原理和实现手段。
- 讲解通俗，步骤详细：在介绍大型实例时，制作步骤都以通俗易懂的语言阐述，并穿插讲解和技巧文字，在阅读时就像听课一样详细而贴切。读者只需要按照步骤操作，即可轻松完成一个实例的创建，不但掌握了开发的步骤，还能掌握开发的技巧。

本书内容

本书具有以下内容：

- 第 1 章主要介绍 Swing 的基本知识，其中包括 Swing 以及其前身 AWT 的背景知识。

- 第2章主要介绍 Java Swing 的开发工具, 通过举例来为读者详细讲解如何通过开发工具来开发 Java Swing 图形界面。
- 第3章主要介绍 Swing 中组件的基本知识。
- 第4章主要介绍有关标签组件和按钮组件的使用方法。
- 第5章主要介绍有关布局管理器的知识, 通过布局管理器来将组件排列在容器内, 从而构成一个完美的界面。
- 第6章主要介绍有关中间容器, 即面板方面的知识。
- 第7章主要介绍如何处理 Swing 事件的发生。
- 第8章主要介绍有关列表框和下拉列表框的使用方法。
- 第9章主要介绍有关进度条组件、时间组件、滑块组件和分隔条组件的知识。
- 第10章主要介绍有关文件选取器和颜色选取器的使用方法。
- 第11章主要介绍文本组件的使用以及各种文本组件的使用环境。
- 第12章主要介绍有关窗口的知识, 其中包括一般窗口、对话框和 JApplet。
- 第13章主要介绍有关工具条和菜单的使用方法。
- 第14章主要介绍有关表格的创建以及使用方法。
- 第15章主要介绍有关树组件的创建以及使用方法。
- 第16章主要介绍有关界面观感器的使用。
- 第17章主要介绍 Java 语言中如何对数据进行处理。
- 第18章主要介绍 Swing 的模型架构, 包括 MVC 和可分离的 MVC。
- 第19章主要介绍平时在开发中经常使用的细微知识点。
- 第20章主要为读者列举一个综合实例——通讯录系统的开发方法和思路。

目标读者

本书具有知识全面、实例精彩、指导性强的特点, 力求以全面的知识及丰富的实例来指导读者掌握 Java Swing 图形编程技术。本书适合以下读者阅读:

- Java 程序语言的初学者。
- 具有一定编程经验的初级编程人员。

资源下载与答疑

可以登录图格新知网站 <http://www.booksaga.com> 下载本书所有实例的源代码。在学习过程中, 遇到疑难问题, 可以通过以下方式与我们联系: booksaga@126.com。

本书由王鹏、何昀峰编著, 此外, 参与本书编写和资料整理的还有彭纯祥、毕利、左中凯、蒋勇、许丹、刘伟、吴军、刘龙、何长亮、刘大勇、陈强、王伟、方杰、潘磊、郑见、徐之祥、宋磊等, 在此, 编者对他们表示衷心的感谢。

编者

2008年9月

目 录

Contents

第 1 章	Java Swing 概述.....	1
1.1	什么是 Java Swing.....	1
1.1.1	Swing 的发展史.....	1
1.1.2	Swing 的功能.....	2
1.1.3	Swing 的特性.....	2
1.2	Java Swing 的包结构.....	3
1.3	一个 Java Swing 程序实例.....	5
1.4	本章小结.....	6
1.5	本章习题.....	6
第 2 章	如何使用 IDE 开发 Swing 程序.....	7
2.1	如何利用 Eclipse 开发 Swing 程序.....	7
2.2	如何利用 JBuilder 开发 Swing 程序.....	11
2.3	如何利用 NetBeans 开发 Swing 程序.....	16
2.4	本章小结.....	19
2.5	本章习题.....	19
第 3 章	Java Swing 组件基础.....	20
3.1	Swing 组件类的层次.....	20
3.2	Window 类.....	22
3.2.1	顶层容器类和包含层次.....	22
3.2.2	在顶层容器中添加组件.....	23
3.2.3	在顶层容器中添加菜单栏.....	25
3.3	JComponent 类.....	27
3.4	本章小结.....	31
3.5	本章习题.....	32

第 4 章 如何使用标签和按钮组件.....	35
4.1 如何使用标签	35
4.2 如何使用按钮	37
4.2.1 如何使用普通按钮	38
4.2.2 如何使用单选按钮	39
4.2.3 如何使用复选框	42
4.2.4 按钮组件的实例应用	43
4.3 本章小结	45
4.4 本章习题	45
第 5 章 如何使用布局管理器组件.....	48
5.1 布局管理器概述	48
5.2 布局管理器的种类	49
5.2.1 BorderLayout.....	49
5.2.2 FlowLayout.....	53
5.2.3 GridLayout	56
5.2.4 GridBagLayout.....	59
5.2.5 CardLayout.....	61
5.2.6 BoxLayout	65
5.2.7 SpringLayout	67
5.2.8 GroupLayout.....	68
5.3 自定义布局管理器的创建	70
5.4 本章小结	70
5.5 本章习题	71
第 6 章 如何使用面板组件	73
6.1 如何使用 JPanel.....	73
6.2 如何使用 JScrollPane	76
6.3 如何使用 JSplitPane	78
6.4 如何使用 JTabbedPane	81
6.5 如何使用 JInternalFrame	84
6.6 如何使用 JLayeredPane.....	86
6.7 如何使用 JRootPane	88
6.8 本章小结	90
6.9 本章习题	90

第 7 章	Swing 事件处理机制	92
7.1	Swing 事件处理机制概述	92
7.2	Swing 中的监听器	93
7.2.1	事件处理的过程与步骤	93
7.2.2	匿名类方式处理事件	94
7.2.3	适配器类	95
7.2.4	Swing 所支持的事件监听器	96
7.2.5	窗口事件的处理	96
7.2.6	动作事件的处理	99
7.2.7	焦点事件的处理	100
7.3	本章小结	102
7.4	本章习题	102
第 8 章	如何使用列表框和下拉列表框组件	105
8.1	如何使用列表框 JList	105
8.1.1	使用数组方式创建列表框	105
8.1.2	使用 Vector 方式创建列表框	106
8.1.3	使用 ListModel 方式创建列表框	108
8.1.4	列表框选取事件的处理	111
8.1.5	列表框双击事件的处理	112
8.2	如何使用下拉列表框 JComboBox	115
8.2.1	使用数组和 Vector 创建下拉列表框	115
8.2.2	使用 ComboBoxModel 创建下拉列表框	116
8.2.3	下拉列表框的事件处理	119
8.3	本章小结	121
8.4	本章习题	121
第 9 章	如何使用进度条、时间、滑块和分隔条组件	123
9.1	如何使用进度条组件 JProgressBar	123
9.2	如何使用时间组件 Timer	125
9.3	如何使用滑块组件 JSlider	127
9.4	如何使用分隔条组件 JSeparator	129
9.5	本章小结	130
9.6	本章习题	130

第 10 章 如何使用选取器组件.....	132
10.1 如何使用文件选取器 JFileChooser	132
10.1.1 如何创建 JFileChooser 组件	133
10.1.2 如何创建 JFileChooser 对话框	138
10.2 如何使用颜色选取器 JColorChooser	141
10.3 本章小结	143
10.4 本章习题	143
第 11 章 如何使用文本组件.....	145
11.1 文本组件概述	145
11.2 如何使用普通文本组件	146
11.2.1 如何使用 JTextField	146
11.2.2 如何使用 JPasswordField	151
11.2.3 如何使用 JFormattedTextField	154
11.3 如何使用文本区组件	155
11.4 如何打印文本组件	167
11.5 本章小结	168
11.6 本章习题	168
第 12 章 如何使用窗口、对话框和 JApplet 组件.....	171
12.1 如何使用窗口组件	171
12.2 如何使用对话框组件	174
12.3 如何使用 JApplet 组件.....	180
12.4 本章小结	182
12.5 本章习题	182
第 13 章 如何使用菜单和工具条组件.....	185
13.1 如何使用菜单组件	185
13.1.1 菜单组件的类层次	185
13.1.2 如何创建菜单	187
13.1.3 如何处理菜单事件	191
13.1.4 如何响应键盘操作	193
13.1.5 如何使用弹出式菜单	196
13.1.6 如何使用菜单项的启用和禁用功能	198
13.1.7 如何创建复选框菜单项	199

13.1.8	如何创建单选按钮菜单项	201
13.1.9	如何定义个性化菜单	202
13.1.10	菜单组件的常用 API	207
13.2	如何使用工具条组件	207
13.2.1	如何创建工具条	208
13.2.2	如何定义个性化工具条	209
13.2.3	工具条组件的常用 API	212
13.3	本章小结	212
13.4	本章习题	212
第 14 章	如何使用表格组件	214
14.1	如何创建一个表格	214
14.2	如何把表格加入容器	217
14.3	如何设置表格列宽	221
14.4	如何创建表格模型	222
14.5	如何监听数据变化	227
14.6	如何使用选择器	229
14.7	如何使用编辑器和渲染器	232
14.8	如何使用自定义渲染器	236
14.9	如何为单元格指定文字说明	238
14.10	如何为表头指定文字说明	239
14.11	如何使用排序和过滤	240
14.12	如何使用组合框作为编辑器	242
14.13	如何使用其他编辑器	243
14.14	如何使用编辑器验证文本	245
14.15	如何打印表格	247
14.16	本章小结	247
14.17	本章习题	247
第 15 章	如何使用树组件	249
15.1	如何创建树	249
15.2	如何创建数据模型	253
15.3	如何处理节点事件	255
15.3.1	如何处理 TreeModelEvent 事件	255
15.3.2	如何处理 TreeSelectionEvent 事件	263

15.4	如何定义个性化树	269
15.5	树组件的常用 API	269
15.6	本章小结	270
15.7	本章习题	270
第 16 章	如何使用 Swing 观感器	273
16.1	如何设置程序的观感	273
16.2	如何自定义观感器	276
16.3	本章小结	281
16.4	本章习题	281
第 17 章	Swing 与并发	282
17.1	多线程问题	282
17.2	初始线程	283
17.3	事件分派线程	284
17.4	工作线程	286
17.4.1	简单的背景任务	287
17.4.2	拥有临时结果的任务	292
17.4.3	取消背景任务	295
17.4.4	绑定属性和状态方法	297
17.5	本章小结	298
17.6	本章习题	298
第 18 章	Swing 模型架构	300
18.1	传统的 MVC 设计模式	300
18.2	可分离的模型架构	302
18.3	本章小结	303
18.4	本章习题	303
第 19 章	Swing 的其他特性	304
19.1	如何在 Swing 组件中使用 HTML	304
19.2	如何使用边框	309
19.2.1	如何使用 Swing 中的边框	309
19.2.2	如何创建自定义边框	314
19.2.3	边框组件的常用 API	315
19.3	如何使用图标	316

19.4 如何使用动作	318
19.5 如何支持辅助技术	321
19.6 如何使用焦点子系统	321
19.7 如何使用键绑定	326
19.8 如何在对话框中使用 Modality	328
19.9 如何创建 Splash Screen	332
19.10 如何使用 System Tray	332
19.11 如何使用 Swing 拖曳功能和数据传输	333
19.12 本章小结	336
19.13 本章习题	336
第 20 章 Swing 实现通讯录系统	338
20.1 通讯录系统的软件框架	338
20.2 通讯录系统的登录系统	339
20.3 通讯录系统的主菜单系统	342
20.3.1 数据库模块的设计	343
20.3.2 信息界面模块的设计	360
20.3.3 功能模块的设计	374
20.3.4 其他模块的设计	394
20.3.5 TabbedPane 容器框架的设计	397
20.3.6 主菜单的设计	398
20.4 本章小结	407

第1章 Java Swing 概述

Java Swing 的技术发展到现在, 已经被许多开发人员作为图形开发的首选。相对于 AWT 技术来说, Java Swing 技术有过之而无不及。在这里将会通过对 Java Swing 发展史、功能、特性等方面的学习, 使读者能够清晰地了解 Java Swing 成为图形开发的主流的原因。应用 Java Swing 进行图形开发时, 最重要的就是要学会熟练地应用 Java Swing 包中提供的各种各样的 API, 从而为以后的学习和开发奠定扎实的基础。

1.1 什么是 Java Swing

Swing 是一个用于开发 Java 图形界面应用程序的开发工具包, 它是以抽象窗口工具包(AWT)为基础, 使跨平台应用程序可以使用任何可插拔的外观风格。Swing 开发人员通过使用少量的代码, 就可以利用 Swing 包中丰富、灵活的功能和模块化组件类来开发出令人满意的用户界面。

1.1.1 Swing 的发展史

由于 Swing 是以 AWT 为基础的, 所以在学习 Swing 之前, 首先必须要了解什么是 AWT, AWT 是 Abstract Window Toolkit (抽象窗口工具集) 的全称, 在 AWT 包中拥有很多组件类, 这些组件类是被用来开发图形程序的基础。而 Swing 则是在 AWT 的基础上对这些组件进行了修改和升级等, 从而避免了在 AWT 开发中遇到的一些问题。AWT 被 Swing 取代, 主要是因为包中的组件类已经不能满足日益增长的客户要求。

其实, AWT 一开始并不是专门为用户界面 (UI) 工具包而设计的, 其初衷是用来开发小的应用程序中的图形界面, 所以针对图形界面开发时, AWT 中的功能很少, 并且很多重要的功能在 AWT 中都不具备, 例如剪贴板、打印支持和键盘导航等特性, AWT 包中甚至没有弹出式菜单或滚动窗格等基本特性, 而这两个功能是目前图形开发中不可缺少的。

另外, AWT 有很高的错误发生率。正是因为这一点, 第三方开始提供他们自己的工具包, 这些工具包的可靠性高, 且比 AWT 具有更多功能的组件构架。这些工具包之一就是 Netscape 的 Internet 基础类 (IFC), IFC 是一组建立在 NEXTSTEP 中的用户界面工具包概念基础上的一组轻量类。IFC 组件不是对等的, 在许多方面胜过了 AWT 组件。IFC 还吸引了更多的开发人员加盟。

由于认识到 Java 领域很可能在标准用户界面工具包问题上出现分裂局面, Java 和 Netscape 达成了一个共识, 即共同实现 Java 基础类。Netscape 开发人员与 Swing 工程师一起合作, 以便把大部分的 IFC 功能嵌入到 Swing 组件中。

起初打算让 Swing 类似于 Netscape 的 IFC。但是随着时间的推移, 在增加了插入式界面

样式等特性并修改设计后, Swing 大大偏离了原来的目标。随着 Swing 1.1 版本的推出, 虽然大量的 IFC 技术仍然嵌在 Swing 中, 但是 Swing 与 IFC 类似的大部分已消失。目前, Swing 可以被称为一个功能全面的用户界面工具包, 它拥有 AWT 和 IFC 中最优秀的部分, 并且非常注重个性化的设计。

1.1.2 Swing 的功能

本节将通过与 AWT 进行比较来详细描述有关 Swing 的一些与众不同的功能。

- AWT 是 Abstract Window Toolkit (抽象窗口工具包) 的缩写, 这个工具包提供了一套与本地图形界面进行交互的接口。AWT 中的图形函数与操作系统所提供的图形函数之间有着一一对应的关系, 也就是说, 当使用 AWT 包来创建图形用户界面时, 实际上就是在使用本地操作系统所提供的图形库来绘制图形界面。
- 由于不同操作系统的图形库所提供的功能是不一样的, 在一个平台上存在的功能在另外一个平台上则可能不存在, 为了实现 Java 语言的“一次编译, 到处运行”的概念, AWT 只能通过牺牲功能来实现其平台无关性, 也就是说, AWT 所提供的图形功能是各种不同操作系统所提供的图形功能的交集。由于 AWT 是依靠本地操作系统的方法来实现其功能的, 所以通常把 AWT 控件称为重量级控件。
- Swing 是在 AWT 的基础上发展起来的一套新的图形界面开发工具, 它提供了 AWT 所能够提供的的所有功能, 并且用纯粹的 Java 代码对 AWT 的功能进行扩充。在实际开发中, 并不是所有的操作系统都能够支持树型控件, 而 Swing 则是利用了 AWT 中所提供的基本作图方法对树型控件进行模拟。由于 Swing 控件是用纯粹的 Java 代码实现的, 因此在一个平台上设计的树型控件可以在其他平台上使用。由于在 Swing 中没有使用本地操作系统的内在方法来实现图形功能, 所以通常把 Swing 控件称为轻量级控件。

正是由于 Swing 的以上功能, 它自然而然地代替了 AWT 成为 Java 图形开发的首选。

1.1.3 Swing 的特性

本节将通过表格的形式为读者一一列出有关 Swing 的宏观特性, 如表 1.1 所示。

表 1.1 Swing 特性一览表

特性	描述
Swing GUI 组件	包括从按钮、分隔窗格到表格的所有组件
可插式外观感觉	允许任何使用 Swing 组件的程序选择其外观感觉
Accessibility (无障碍功能) API	支持辅助技术, 例如屏幕阅读器和点字显示器, 帮助用户获得信息
Java 2D API	允许开发人员在应用程序和 Applet 中方便地使用 2D 图形、文本以及图像
拖放支持	为 Java 应用程序和本机应用程序之间提供了拖放功能
国际化	允许创建与全世界使用不同语言 and 不同文化习俗的用户进行交互的应用程序。通过输入法的架构, 开发人员可以创建能够接收各种不同字符的语言文字的应用程序

正是因为 Swing 拥有以上显著的特性,使得开发人员越来越热衷于使用 Swing 进行图形开发。

1.2 Java Swing 的包结构

Java Swing 由许多包组成,下面将通过表的形式列出这些包及其功能,说明如表 1.2 所示。

表 1.2 Java Swing 包结构及其功能

包名称	功能简介
javax.swing	Swing 组件和实用工具
javax.swing.border	Swing 轻量组件的边框
javax.swing.colorchooser	JColorChooser 的支持类(接口)
javax.swing.event	事件和侦听器类
javax.swing.filechooser	JFileChooser 的支持类(接口)
javax.swing.pending	未完全实现的 Swing 组件
javax.swing.plaf	抽象类,用于定义 UI 代表的行为
javax.swing.plaf.basic	实现所有标准界面样式公共功能的基类
javax.swing.plaf.metal	用户界面代表类,用于实现 Metal 界面样式
javax.swing.table.JTable	组件的支持类
javax.swing.text	支持文档的显示和编辑
javax.swing.text.html	支持显示和编辑 HTML 文件
javax.swing.text.html.parser.html	文件的分析器类
javax.swing.text.rtf	支持显示和编辑 RTF 文件
javax.swing.tree.JTree	组件的支持类
javax.swing.undo	支持取消操作

下面将针对常用包给予详细说明。

- swing 包是 Swing 提供的最大包,包含将近 100 个类和 25 个接口。几乎所有的 Swing 组件都在 swing 包中,只有 JTableHeader 和 JTextComponent 例外,它们分别存放在 swing.table 包和 swing.text 包中。
- swing.border 包中含有多个在轻量 Swing 组件的边衬中用于绘制边框的类。border 包由一个 Border 接口、一个 AbstractBorder 类和 AbstractBorder 的许多具体扩展组成。
- swing.event 包中定义了事件和事件监听器类,swing.event 包与 AWT 的 event 包类似。awt.event 和 swing.event 都包含事件类和监听器接口,它们分别响应由 AWT 组件和 Swing 组件激发的事件,例如当在树组件中需要节点扩展(或折叠)的通知时,则要实现 Swing 的 TreeExpansionListener 接口,并把一个 TreeExpansionEvent 实例传送给 TreeExpansionListener 接口中定义的方法。TreeExpansionListener 和 TreeExpansionEvent 都是在 swing.event 包中定义的。

- Swing 包含 250 多个类，是组件和支持类的集合。Swing 提供了 40 多个组件，是 AWT 组件的 4 倍。除提供替代 AWT 重量组件的轻量组件外，Swing 还提供了大量有助于开发图形用户界面的附加组件。

1.3 一个 Java Swing 程序实例

上节为读者详细介绍了有关 Java Swing 包的结构,并且针对每个包讲述了其功能,本节将通过一个实例进行讲解,从而使读者能够更加轻松地学习本书后面的知识。

以下实例用于显示一个带有 helloswing 标题的窗口,示例代码如下:

```
import javax.swing.JFrame;
public class Swingtest
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("helloSwing");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
}
```

上述代码的运行结果如图 1.1 所示。

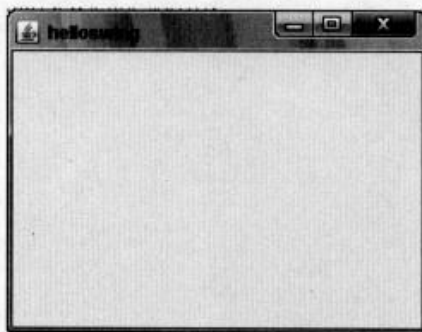


图 1.1 helloswing 程序代码的运行结果

为了能够让读者更加清晰地了解此实例,下面将给予详细分析。

- `import javax.swing.JFrame`: 是一个包的导入代码,因为在程序代码中需要使用到这个包中的类以及其中的方法。其实在大多数的程序代码中还需要引入两个 AWT 的包,一个是“`Java.awt.*`”,另一个是“`Java.awt.event.*`”,这些包都是必须的,因为 Swing 组件使用了 AWT 的基本架构,这两个包中包括了 AWT 的事件模型。
- `jf.setSize(WIDTH,HEIGHT)`: 代码的含义是设置窗口的大小。
- `jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: 代码的含义是使窗口上的最大化、最小化以及关闭键发挥作用。
- `jf.setVisible(true)`: 代码的含义主要是让前面创建的窗口显示出来。

上面编写的窗口也被称为顶层窗口,编写好顶层窗口后就可以在顶层窗口中添加各种各样的组件,从而实现各种各样的功能。有关窗口方面的知识,本书在后面的章节中会有详细讲解,这里不再赘述。

1.4 本章小结

本章通过对 Java Swing 的发展史的介绍,使得读者能够有一个基本的背景知识。通过对 Java Swing 功能的介绍,使得读者能够了解 Java Swing 的用处所在。最后通过分析一个简单的实例,使读者能够对 Java Swing 图形开发有一个简单地了解。

从下一章开始,将具体引导读者如何利用 Java Swing 图形开发工具包开发出复杂实用的应用程序。

1.5 本章习题

1. 简单叙述一下 Java Swing 包和 AWT 包的不同点?
2. 请简单叙述有关包的概念?
3. Java Swing 包是 AWT 包的升级,那么是否可以完全不使用 AWT 包中的类?
4. 在 Java 图形开发时,为什么需要引用各种各样的包呢?



第2章 如何使用IDE开发Swing程序

本章将主要讲述 Eclipse、JBuilder、NetBeans 这三种开发工具的基本使用方法，同时这三种 IDE 开发工具也是目前比较流行的开发工具。IDE 环境就是集成开发环境，集成开发环境的英文是 Integrated Development Environment。过去编写程序是利用文本编辑软件编写源程序，利用编译程序对其进行编译而生成 OBJ 文件（目标文件），再用连接程序将 OBJ 文件与库文件（LIB 文件）连接，从而生成可执行的 EXE 文件，而后还要对程序进行调试。整个过程很复杂。但 IDE 通常集编辑、编译、连接、调试为一体，使得程序的开发变得很方便。Turbo C、Turbo Pascal、Borland C++、Visual C++、Delphi 等都包含有很好的 IDE 在里面，通常说的用它们编写程序，实际上都是在它们的 IDE 中进行工作。

在介绍如何使用这些开发工具开发 Swing 图形程序前，将分别为读者介绍 Eclipse、JBuilder、NetBeans 三种开发工具的一些基本知识，并且通过简单的图形开发程序，分别使用此三种开发工具进行编写、编译、运行。

2.1 如何利用 Eclipse 开发 Swing 程序

2001 年 11 月，IBM 宣布捐出 4 千万美金用来开发 Eclipse 项目。如此受青睐的 Eclipse 是什么样子？如何使用它呢？本节会使读者对 Eclipse 有一个初步的认识。虽然，目前 Eclipse 项目还没有最后完成，但从已有的版本中应该能体会到 Eclipse 设计的主导思想和主要功能特点。其实，Eclipse 就是一个图形化开发 Java 等程序的工具。

当打开 Eclipse 的界面时会出现如图 2.1 所示的界面。

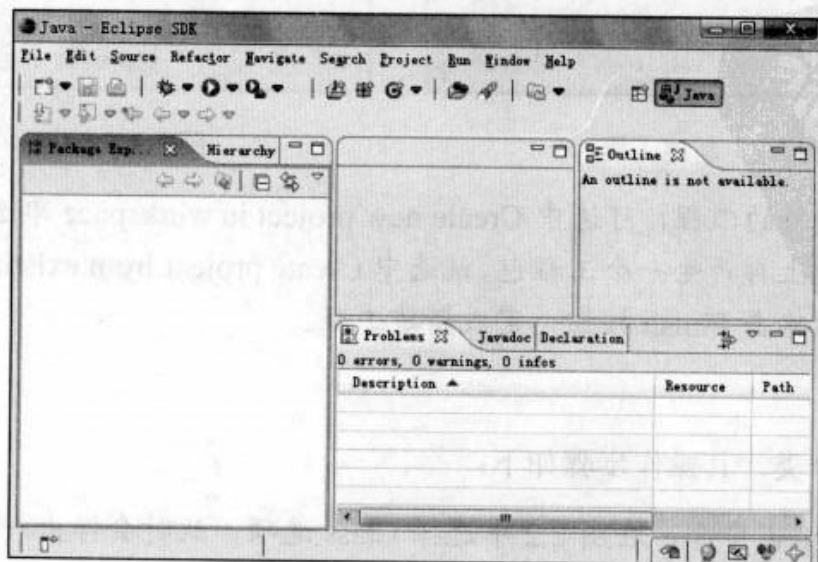


图 2.1 Eclipse 软件界面

为了能够让读者了解如何使用 Eclipse 开发工具开发 Swing 图形界面程序，在这里将通过实例来讲述使用这种开发工具开发一个图形化程序的步骤。

1. 创建 Java 工程

首先来创建一个 Java 工程，其步骤如下。

01 打开 Eclipse 应用程序，选择 File | New 命令，弹出如图 2.2 所示的对话框。

提示



工程在整个程序中作为一个项目组，其相当于一个成形的软件。在这个工程中包含有类、包、接口等元素，把这些元素组合在一起打包成一个应用软件。在编写程序之前，必须要创建一个工程，否则一切都无法开始。工程的创建就好比是一个应用程序框架的搭建，只有先创建了工程，才能往这个工程的框架里放入各种各样的元素。

02 在下拉列表中选择 Java Project 选项后，输入这个新建工程的名称，单击 Next 按钮弹出如图 2.3 所示对话框。

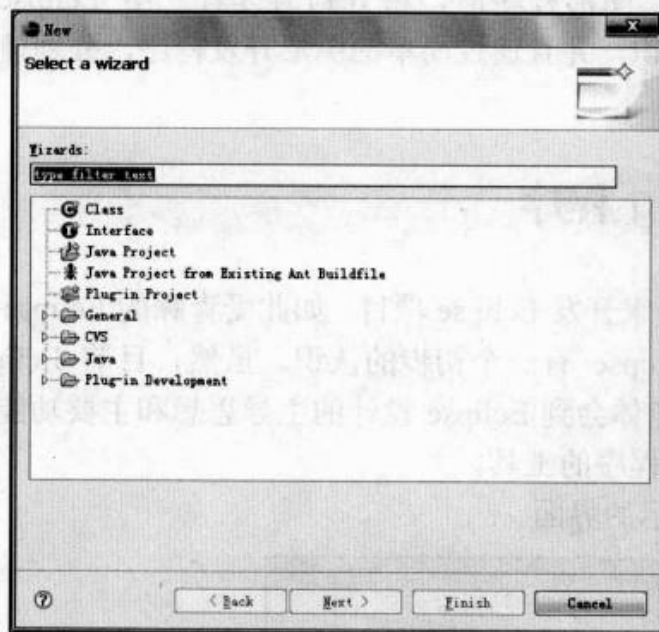


图 2.2 新建工程

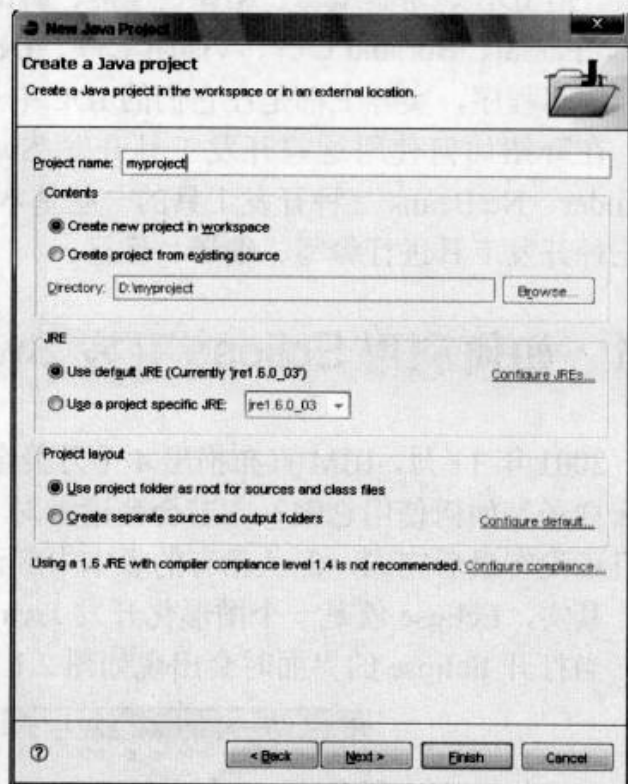


图 2.3 产生一个新的工程

03 如果要产生新的工程，可选中 Create new project in workspace 单选按钮，如果在原来工程的基础上再产生一个工程包，就选中 Create project from existing source 单选按钮。选择完毕后单击 Finish 按钮，完成新建工程。

2. 创建类

接下来创建一个类，其操作步骤如下：

01 选择 File | New 命令，在图 2.2 中选择 Class 选项，此时会弹出一个新建类的对话框，如图 2.4 所示。

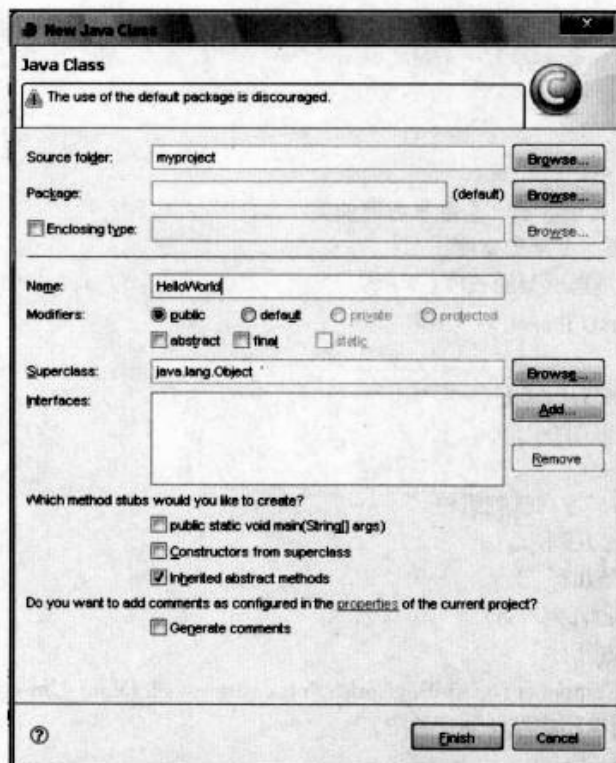


图 2.4 新建一个类

02 在图 2.4 中输入一个类的名称后，单击 Finish 按钮，弹出一个代码编辑框，如图 2.5 所示。

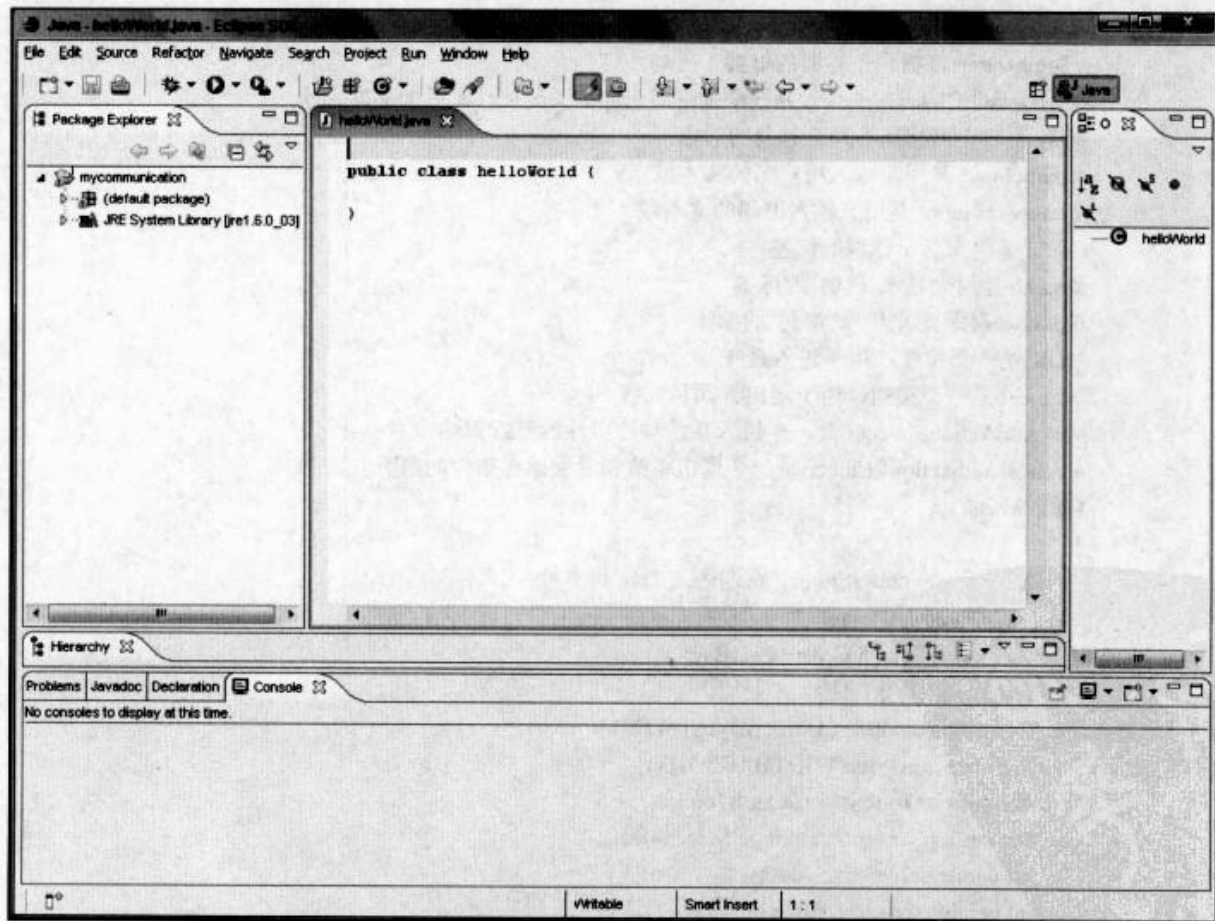


图 2.5 编辑程序代码

03 在编辑窗口中输入如下代码：

// 这段代码主要是创建一个登录窗口界面, 在这个界面中有文本组件、普通按钮组件、标签组件
// 它们是按照网格组布局管理方式布局

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

// 这是一个登录类, 可设计成一个继承容器的类

// WIDTH 是指整个顶层框架的宽度

// HEIGHT 是指整个顶层框架的长度

```
class HelloWorld extends JPanel
```

```
{
```

```
    static final int WIDTH=300;
```

```
    static final int HEIGHT=150;
```

```
    JFrame loginframe;
```

// 按照网格组布局方式排列组件

// x 指控件位于第几列

// y 指控件位于第几行

// w 指控件需要占几列

// h 指控件需要占几行

```
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
```

```
    { // 此方法用来添加控件到容器中
```

```
        constraints.gridx=x;
```

```
        constraints.gridy=y;
```

```
        constraints.gridwidth=w;
```

```
        constraints.gridheight=h;
```

```
        add(c,constraints);
```

```
    }
```

// 这是一个构造器方法

// loginframe 是指这个界面的框架

// setDefaultCloseOperation 是一个使窗口上面的关闭控件有效的类库方法

// lay 是一个网格组布局管理器的对象

// nameinput 是用来输入用户名的文本域

// passwordinput 是用来输入密码的文本域

// title 是用来显示标题的标签

// name 是用来显示"姓名"的标签

// password 是用来显示"密码"的标签

// OK 是一个按钮, 用于进入系统

// cancel 是一个按钮, 用于退出界面和系统

// ok.addActionListener 是一个进入系统动作事件的监听方法

// cancel.addActionListener 是一个退出系统和界面动作事件的监听方法

```
    HelloWorld()
```

```
    {
```

```
        loginframe=new JFrame("欢迎进入 Java 世界");
```

```
        loginframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        GridBagLayout lay=new GridBagLayout();
```

```
        setLayout(lay);
```

```
        loginframe.add(this, BorderLayout.WEST);
```

```
        loginframe.setSize(WIDTH,HEIGHT);
```

```
        Toolkit kit=Toolkit.getDefaultToolkit();
```

```
        Dimension screenSize=kit.getScreenSize();
```

```
        int width=screenSize.width;
```

```
        int height=screenSize.height;
```

```
        int x=(width-WIDTH)/2;
```

```
        int y=(height-HEIGHT)/2;
```

```
        loginframe.setLocation(x,y);
```

```
        JButton ok=new JButton("登录");
```



```
 JButton cancel=new JButton("放弃");
 JLabel title=new JLabel("欢迎进入 Java 世界");
 JLabel name=new JLabel("用户名");
 JLabel password=new JLabel("密码");
 final JTextField nameinput=new JTextField(15);
 final JTextField passwordinput=new JTextField(15);
 GridBagConstraints constraints=new GridBagConstraints();
 constraints.fill=GridBagConstraints.NONE;
 constraints.anchor=GridBagConstraints.EAST;
 constraints.weightx=3;
 constraints.weighty=4;
 add(title,constraints,0,0,4,1);           // 使用网格组布局添加控件
 add(name,constraints,0,1,1,1);
 add(password,constraints,0,2,1,1);
 add(nameinput,constraints,2,1,1,1);
 add(passwordinput,constraints,2,2,1,1);
 add(ok,constraints,0,3,1,1);
 add(cancel,constraints,2,3,1,1);
 loginframe.setResizable(false);
 loginframe.setVisible(true);
 }
 public static void main(String[] args)
 {   HelloWorld hello=new HelloWorld();
 }
```

上面的 Swing 程序代码的运行结果如图 2.6 所示。

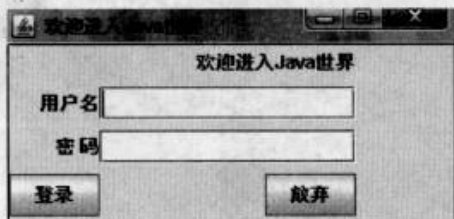


图 2.6 程序运行结果

通过对这个开发工具的讲解，相信读者已经对它有了基本的了解，由于篇幅所限，这里不再过多详解，希望读者以本节为基础，继续深入学习。

2.2 如何利用 JBuilder 开发 Swing 程序

本节将为读者讲述如何使用 JBuilder 工具进行 Swing 程序开发，在讲述 JBuilder 开发工具之前，首先为读者简单介绍一下 JBuilder 开发工具的基本知识。

JBuilder 工具是 Borland 公司开发的一个跨平台的 Java 开发环境，可以用来开发符合业界标准的 Java 应用系统。本节将以 JBuilder 9.0 为例，为读者讲述其特性和使用方法。

JBuilder 9.0 是目前比较新、比较成熟的版本，其拥有许多重要的特性：

- UML 可视化展现代码程序。
- 高效率的 Web 服务开发环境。
- 单元测试功能。

- 程序代码重构功能。
- 简单的程序发布功能。
- 支持团队开发机制。

正是由于以上特点,才能让 JBuilder 工具真正得到开发人员的青睐。JBuilder 9.0 同 Eclipse 一样也有不同的使用版本,一般来说分为三种,其详细内容如下。

- JBuilder 9.0 Enterprise (企业版): 它适合开发各种 Java 应用程序,功能包括程序代码的编写、调试和部署等,可以协助开发人员开发各种企业级标准的企业级应用系统。
- JBuilder 9.0 Developer (开发版): 在它的内部拥有各种高效率的辅助开发工具,并提供了团队开发所需要的版本管理功能,从而为团队开发提供了方便。
- JBuilder 9.0 Personal (个人版): 针对学习 Java 程序的学生和个人,为其提供一个学习 Java 的开发环境。

其实每个版本的基本用法是一样的,只不过其中的外挂包不一样,JBuilder 9.0 个人版开发工具的界面如图 2.7 所示。

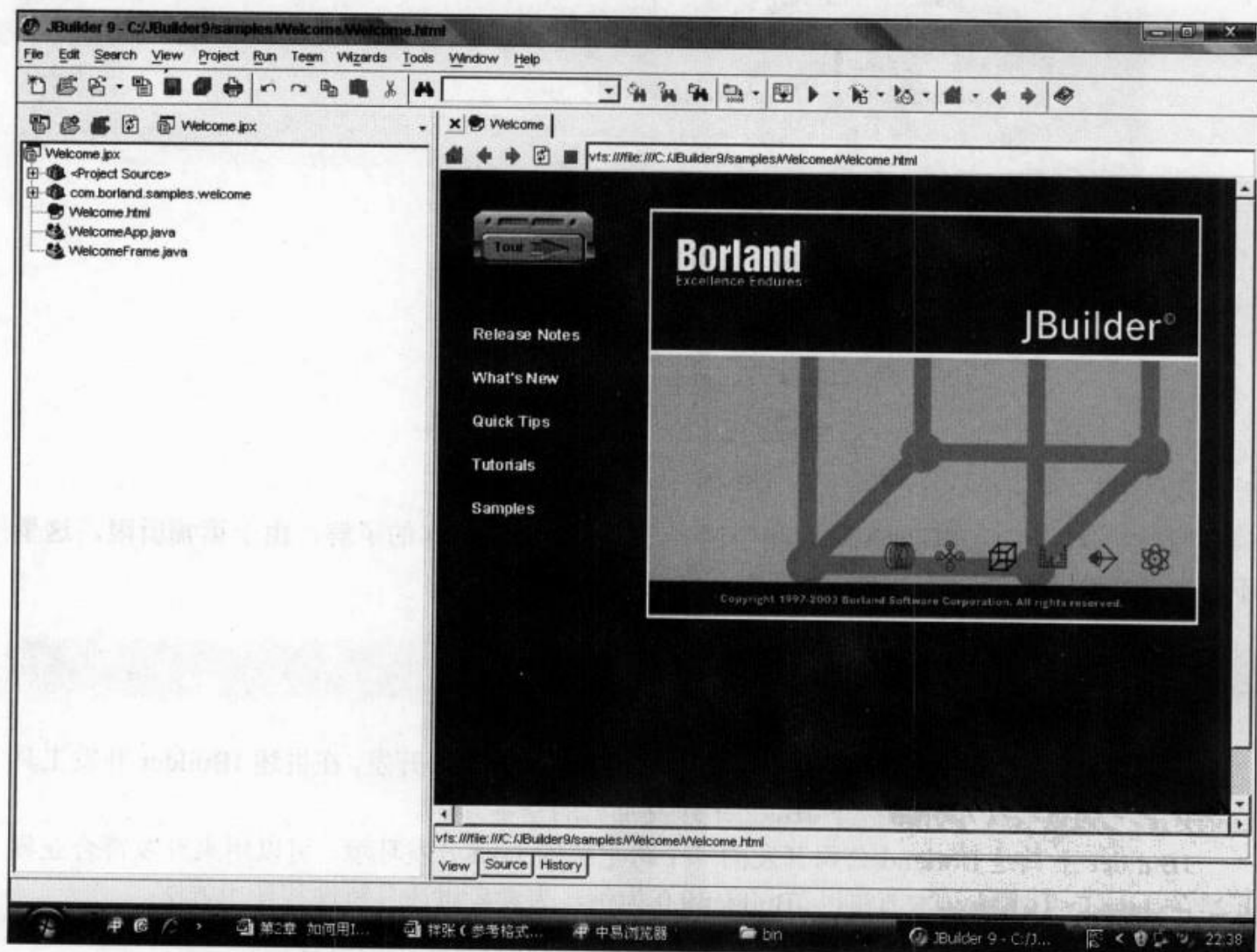


图 2.7 JBuilder 9.0 开发工具界面

接下来将使用 JBuilder 开发工具来编写 HelloWorld.java 程序,操作步骤如下:

01 选择 File | New 命令,弹出 Object Gallery 对话框,如图 2.8 所示。

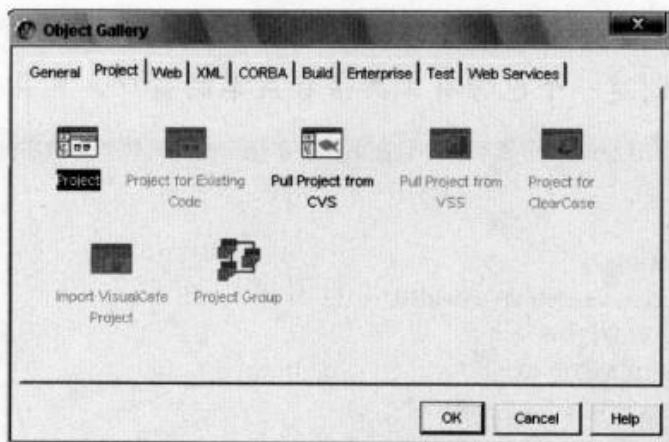


图 2.8 新建工程界面

- 02 在图 2.8 中选择其中的 Project 选项，单击 OK 按钮，弹出如图 2.9 所示的对话框。在 Name 文本框中输入工程的名称。

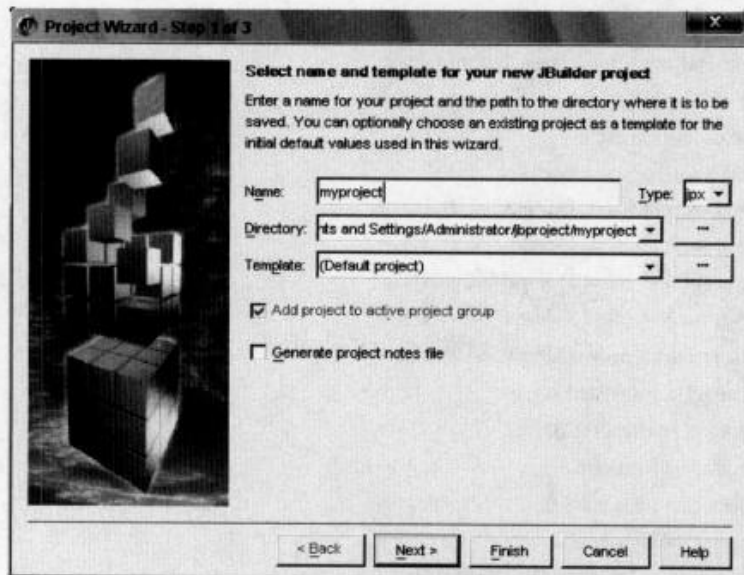


图 2.9 完成创建工程

- 03 单击 Finish 按钮后即可完成新建工程操作。新建工程后还需要在新建的工程中新建一个类，其类的名称为 HelloWorld.java，如图 2.10 所示。

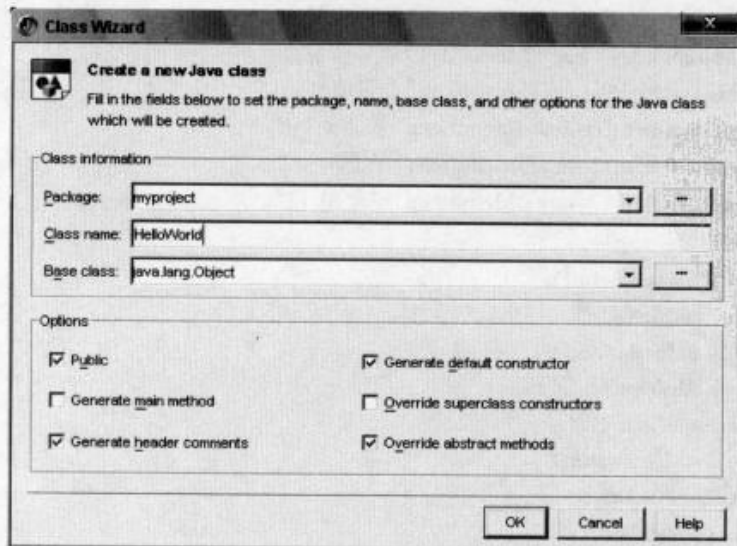


图 2.10 新建类

04 单击 OK 按钮系统会弹出一个编辑区。此时可以开始编辑程序了。下面将列举一个实例，该实例用于创建一个包含有工具条和菜单的窗口，其具体程序代码如下：

// 这段代码主要用于创建一个菜单，并且在菜单下创建三个工具按钮，放在同一个工具条中

```
import javax.swing.*;
import java.awt.*;

public class HelloWorld
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=400;
    JPopupMenu pop;
    JMenuItem item2;
    JFrame f;
    JMenuItem item1;
    JPanel p;
    JToolBar bar;
    public HelloWorld()
    {
        f=new JFrame("我的测试界面");
        JMenuBar menubar1=new JMenuBar();
        p=new JPanel();
        f.setContentPane(p);
        f.setJMenuBar(menubar1);
        JMenu menu1=new JMenu("菜单 1");
        JMenu menu2=new JMenu("菜单 2");
        JMenu menu3=new JMenu("菜单 3");
        JMenu menu4=new JMenu("菜单 4");
        JMenu menu5=new JMenu("菜单 5");
        menubar1.add(menu1);
        menubar1.add(menu2);
        menubar1.add(menu3);
        menubar1.add(menu4);
        menubar1.add(menu5);
        item1=new JMenuItem("子菜单 1");
        item2=new JMenuItem("子菜单 2");
        JMenuItem item3=new JMenuItem("子菜单 3");
        JMenuItem item4=new JMenuItem("子菜单 4");
        JMenuItem item5=new JMenuItem("子菜单 5");
        JMenuItem item6=new JMenuItem("子菜单 6");
        JMenuItem item7=new JMenuItem("子菜单 7");
        JMenuItem item8=new JMenuItem("子菜单 8");
        JMenuItem item9=new JMenuItem("子菜单 9");
        JMenuItem item10=new JMenuItem("子菜单 10");
        JMenuItem item11=new JMenuItem("子菜单 11");
        JMenuItem item12=new JMenuItem("子菜单 12");
        menu1.add(item1);
        menu1.addSeparator();
        menu1.add(item2);
        menu1.addSeparator();
        menu1.add(item3);
        menu2.add(item4);
        menu2.addSeparator();
        menu2.add(item5);
        menu3.add(item6);
        menu3.addSeparator();
        menu3.add(item7);
    }
}
```



```
menu4.add(item8);
menu4.addSeparator();
menu4.add(item9);
menu4.addSeparator();
menu4.add(item10);
menu5.add(item11);
menu5.addSeparator();
menu5.add(item12);
JButton button1 = new JButton("工具 1");
JButton button2 = new JButton("工具 2");
JButton button3 = new JButton("工具 3");
bar = new JToolBar();
bar.add(button1);
bar.add(button2);
bar.add(button3);
BorderLayout bord = new BorderLayout();
p.setLayout(bord);
p.add("North", bar);
f.setVisible(true);
f.setSize(WIDTH, HEIGHT);
Toolkit kit = Toolkit.getDefaultToolkit();
Dimension screenSize = kit.getScreenSize();
int width = screenSize.width;
int height = screenSize.height;
int x = (width - WIDTH) / 2;
int y = (height - HEIGHT) / 2;
f.setLocation(x, y);
}
public static void main(String[] args)
{
    new HelloWorld();
}
```

上述代码的执行结果如图 2.11 所示。

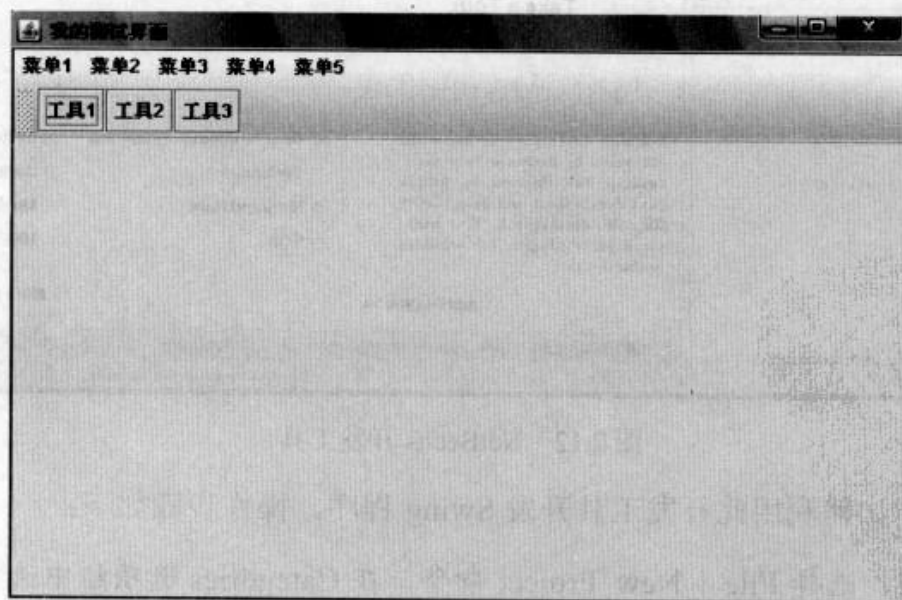


图 2.11 运行结果

其实 JBuilder 的使用同 Eclipse 的用法差不多, 希望读者能够多多练习, 从而能够熟悉它的使用方法, 选择一个好的并且适合自己的开发工具对于开发人员来说, 是尤为重要的, 也希望读者能好好的体会。

2.3 如何利用 NetBeans 开发 Swing 程序

NetBeans 6.0 既可以作为压缩文件的形式下载, 也可以跨平台进行安装。在 Windows 平台下, 安装 NetBeans 后会与操作系统无缝的集成起来, 包括桌面的快捷方式和增加安装、卸载控制面板等。

在 Windows 系统中, NetBeans 的使用非常方便, NetBeans 的菜单布局也非常有逻辑, 使用起来很简单, 大多数的功能很容易上手, 一直以来, NetBeans 在稳定性方面做的十分出色。

使用 NetBeans 可以充分地利用其强有力的 RCP (Rich Client Platform) 向导来创建新的、成熟的 Swing 应用。

NetBeans 开发工具的界面如图 2.12 所示。

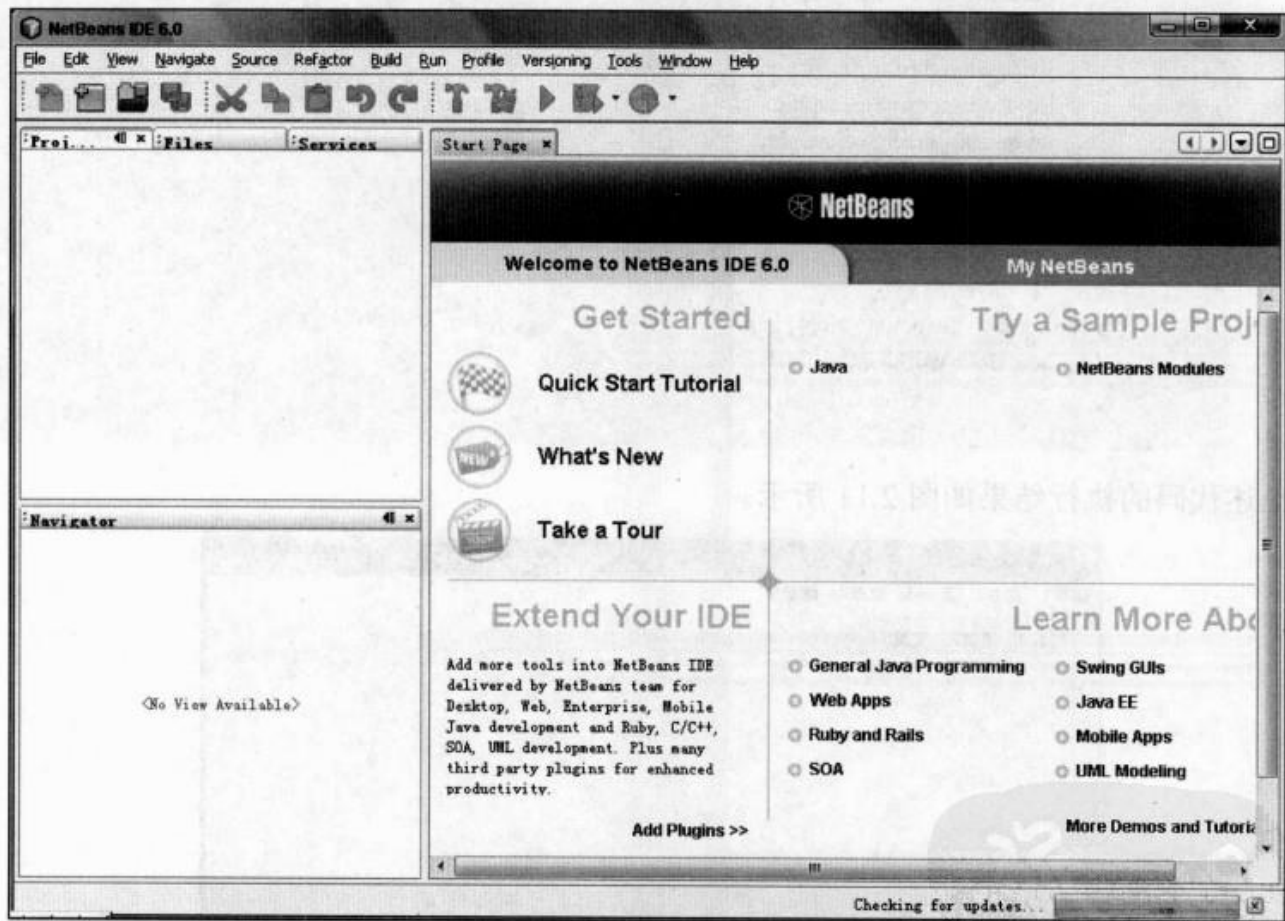


图 2.12 NetBeans 开发工具

接下来将介绍如何利用此开发工具开发 Swing 程序, 操作步骤如下:

- 01 新建工程, 选择 File | New Project 命令, 在 Categories 选项组中选中的 Java 选项, 在 Projects 选项组中选中的 Java Application 选项, 如图 2.13 所示。

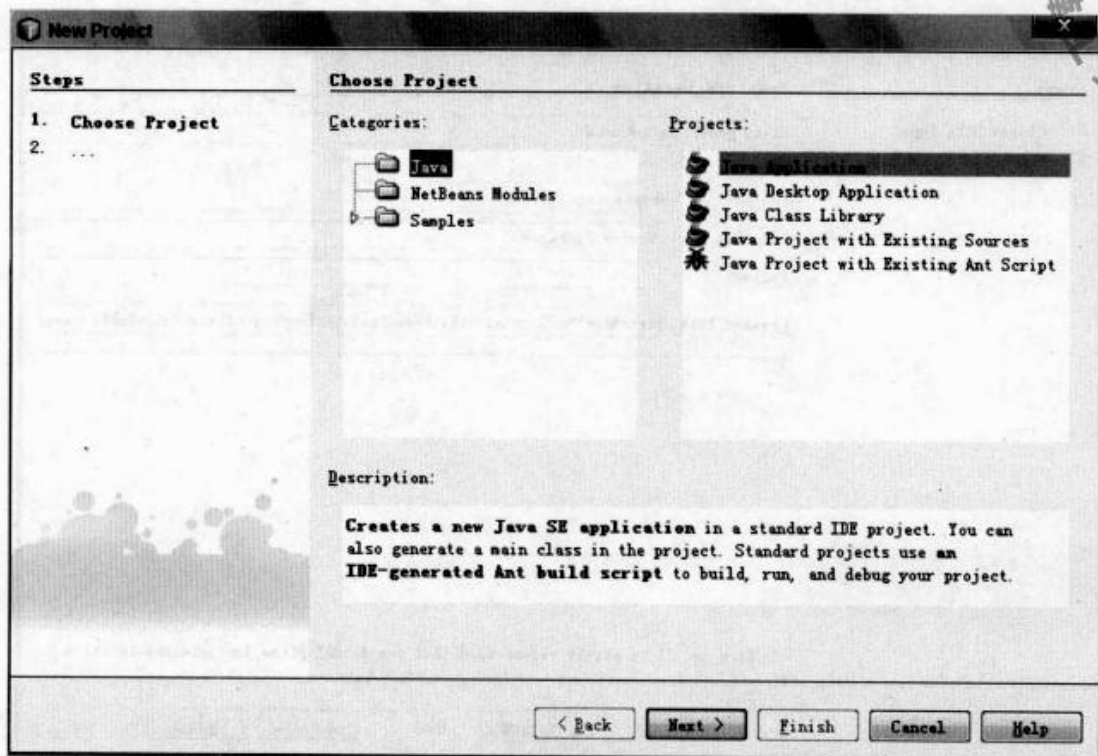


图 2.13 新建工程

02 单击 Next 按钮，弹出如图 2.14 所示的对话框。

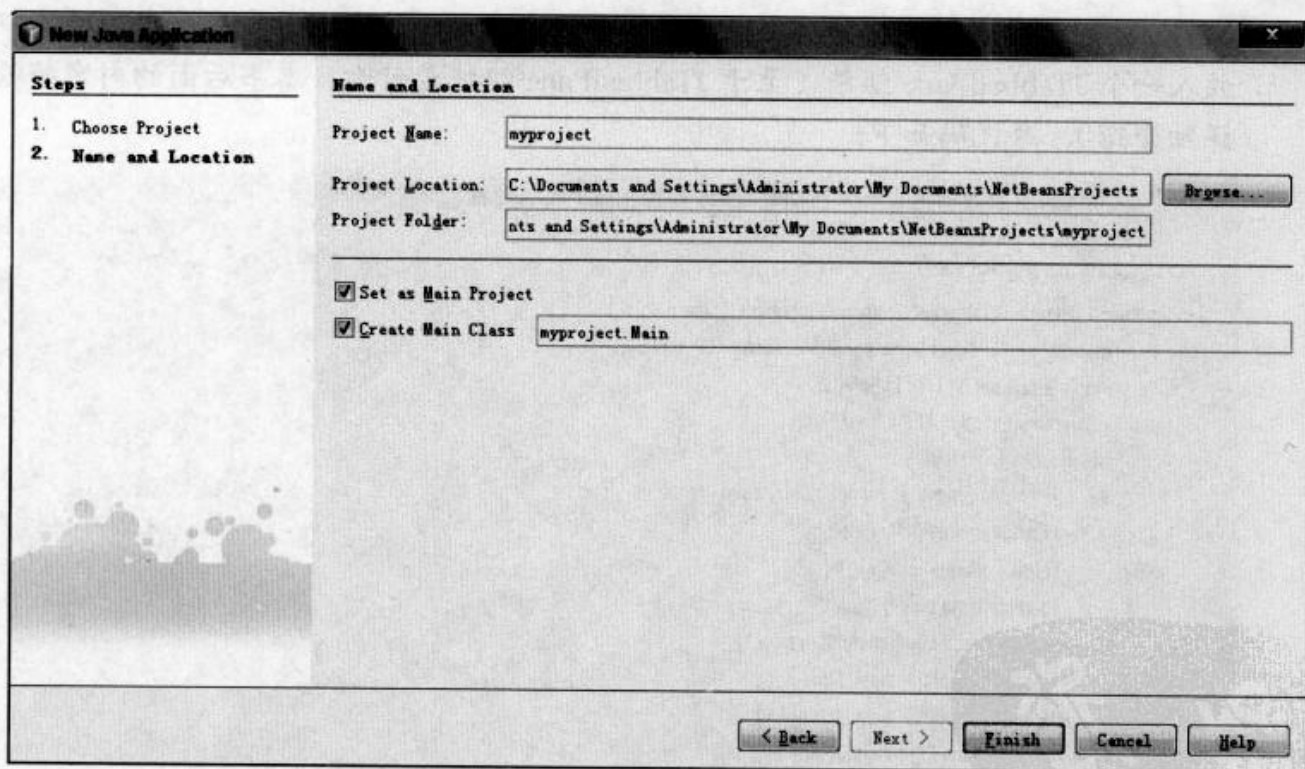


图 2.14 完成新建工程

03 输入工程名称后，单击 Finish 按钮，从而完成新建工程的操作。而后将新建一个 Java 类程序，选择 File | New Java Class 命令，弹出如图 2.15 所示的对话框。

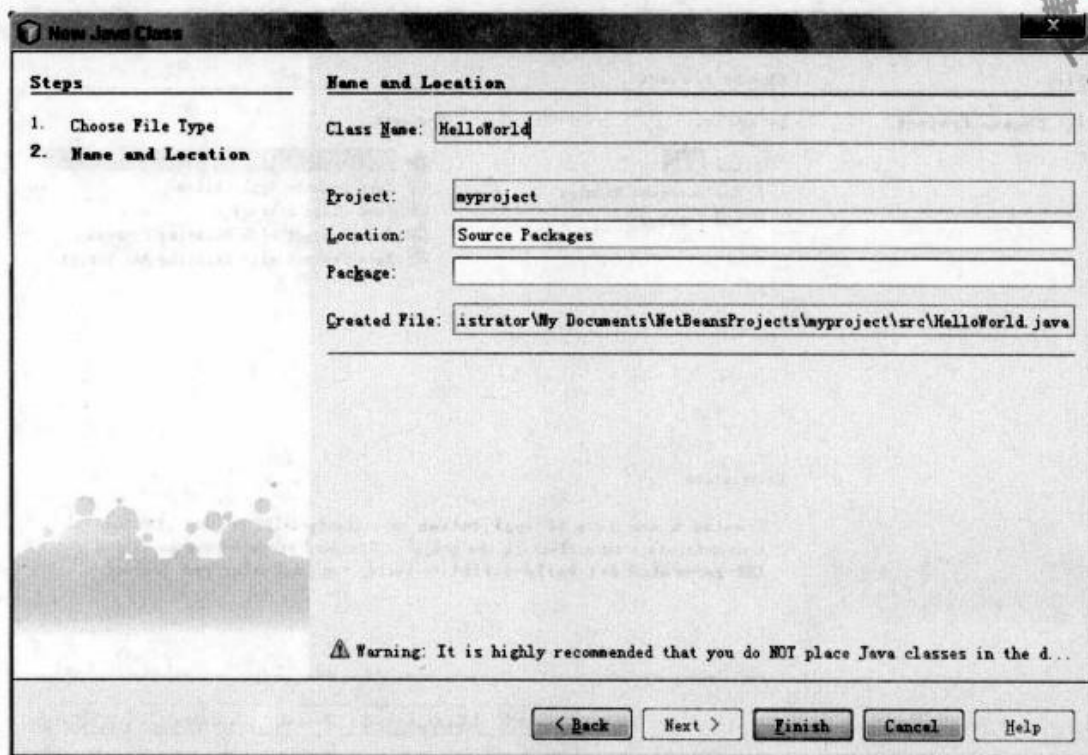


图 2.15 新建类

04 在 Class Name 文本框中输入类名 HelloWorld，单击 Finish 按钮，从而完成新建类的操作。此时，会出现编辑窗口，下面将列举一个实例，该实例用于在一个顶层窗口中放入一个 JTabbedPane 组件（至于 JTabbedPane 组件是什么，本书后面的内容将给予详细介绍），源代码如下：

```
// 这段代码主要是创建了一个 JTabbedPane 面板，这个面板也就是标签页面板
import javax.swing.*;
import java.awt.*;
public class HelloWorld extends JTabbedPane
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=400;
    public HelloWorld()
    {
        JFrame f=new JFrame("测试窗口");
        JPanel p=new JPanel();
        f.setContentPane(p);
        f.setVisible(true);
        setLayout(new BorderLayout());
        JPanel panel1 = new JPanel ();
        JPanel panel2 = new JPanel ();
        JPanel panel3 = new JPanel ();
        JPanel panel4 = new JPanel ();
        JPanel panel5 = new JPanel ();
        panel1.setLayout(new BorderLayout());
        addTab("panel1", panel1);
        setEnabledAt(0,true);
        setTitleAt(0,"测试页 1");
        addTab("panel2", panel2);
        setEnabledAt(1, true);
        setTitleAt(1,"测试页 2");
        addTab("panel3", panel3);
```



```
setEnabledAt(2, true);
setTitleAt(2, "测试页 3");
addTab("panel4", panel4);
setEnabledAt(0, true);
setTitleAt(3, "测试页 4");
addTab("panel5", panel5);
setEnabledAt(4, true);
setTitleAt(4, "测试页 6");
setPreferredSize(new Dimension(500, 200));
setTabPlacement(JTabbedPane.TOP);
setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
p.add("Center", this);
setVisible(true);
}

public static void main(String[] args)
{
    new HelloWorld();
}
```

编辑完程序代码后，单击工具栏中的 Run 按钮，会出现如图 2.16 所示的运行结果。

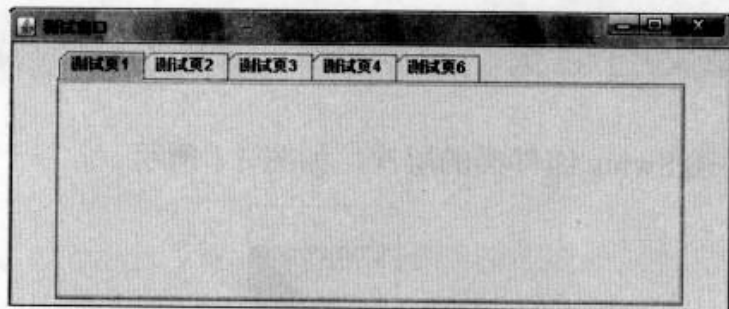


图 2.16 程序运行结果

2.4 本章小结

本章所有的程序代码读者都无须去关心，因为这些代码涉及到了后面需要讲解的知识，本章的关键是学习如何使用这三个开发工具。由上面的操作可以看出，无论是哪一个开发工具，其开发程序的基本操作都是一样的，有兴趣的读者也可以查阅具体的相关资料，或是通过自己的练习发现它们的差异之处，这里不再赘述。

2.5 本章习题

1. 简单叙述一下什么是 IDE 环境？
2. 观察本章所有的 IDE 开发工具，请总结一下它们的共同点？
3. 通过本章的学习，请读者针对 Eclipse 与 JBuilder 两个开发工具，比较一下哪个好？
4. 在实际开发中，JBuilder 有哪几种版本？

第3章 Java Swing 组件基础

所谓的 Java Swing 组件，也就是可以使用它来组成一个图形化界面，例如按钮、标签、树、表格以及框架等，而每个组件又会拥有不同的形态来满足不同用户的需要。Java Swing 组件是构成图形化界面的最基本元素。本章主要是为读者讲述 Java Swing 组件的一些基础知识，包括 Swing 组件类的继承、顶层类的结构、顶层容器类的使用、内容面板的使用、如何在顶层容器中添加菜单栏，以及 JComponent 类的相关知识。学习本章的主要目的是让读者能够充分熟悉整个 Java Swing 组件类的框架。

3.1 Swing 组件类的层次

本节主要为读者讲述 Swing 组件类的层次，如图 3.1 所示。

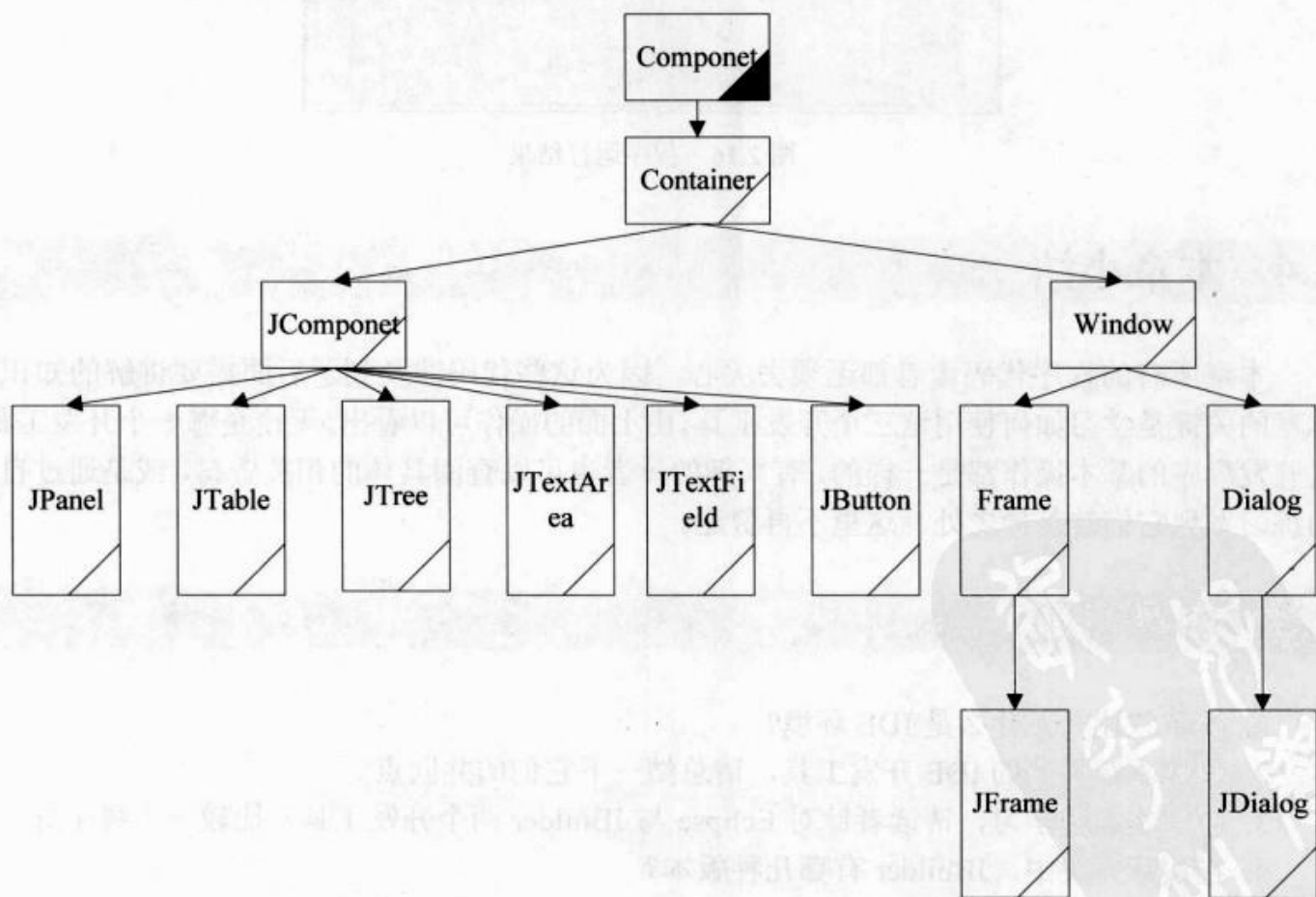


图 3.1 Swing 组件类的层次结构图

从以上的结构示意图中可以知道, Swing 组件可以分成两种类型, 一种是 JComponent 类, 另一种是 Window 类。其中 Window 组件类主要包括了一些可以独立显示的组件, 而 JComponent 组件类主要包括了一些不能独立显示的组件。什么是可以独立显示的组件呢? 可以独立显示的组件就是当运行一个程序时, 这个组件无须托付在其他组件上就可以显示, 即它可以独立显示出来, 例如 JFrame 类, 它可以独立显示, 无须任何其他的依靠组件。而不可独立显示的组件则必须依靠可独立显示的组件来显示, 例如文本框组件、按钮组件必须要依托在 JFrame 框架组件上才能显示出来。相信在后面的章节实例中, 大家一定可以体会到这一点。

以上的结构图只是一个简略的示意图, 实际的结构要比上图复杂的多, 随着后面的学习, 读者会一步一步地掌握 Swing 组件中的各种类。

以上的示意图只是从显示效果上划分的, 其实从其功能上划分, 在 javax.swing 包中 Swing 组件共有三种类型: 顶层组件、中间组件、基本组件。顶层组件又被称作顶层容器, 而中间组件又分为中间容器和特殊中间组件, 如图 3.2 所示。

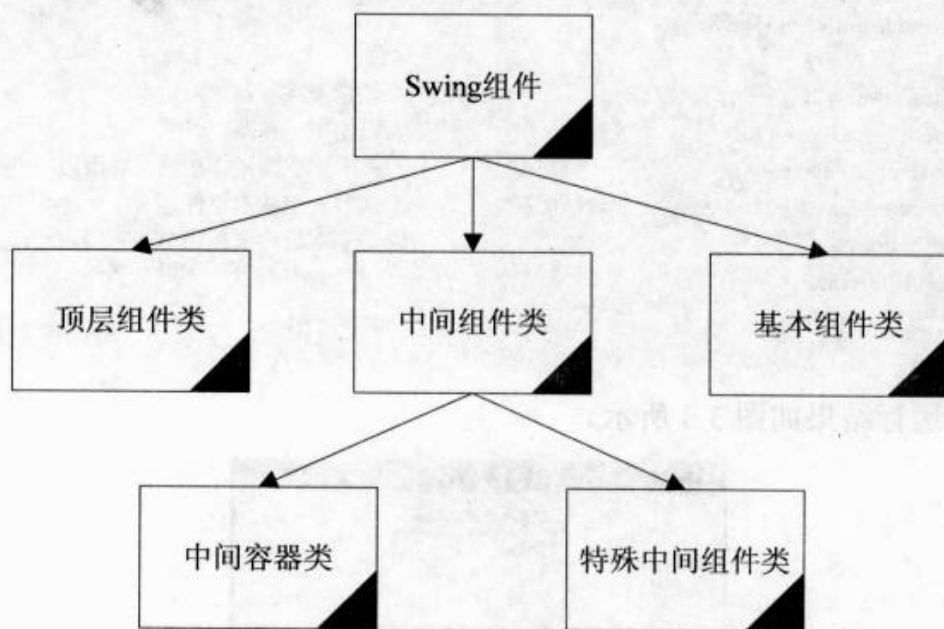


图 3.2 Swing 组件从功能上划分

下面将为读者详细介绍上图中组件类型的含义。

- 顶层容器: JFrame、JApplet、JDialog、JWindow。所谓的顶层容器也可以说是前面所说的 Window 组件, 它是可以独立显示的组件。
- 中间容器: JPanel、JScrollPane、JSplitPane、JToolBar。所谓的中间容器也就是指那些可以充当载体, 但也是不可独立显示的组件。通俗地说, 就是一些基本控件可以放在其中, 但是它不能独立显示, 必须要依托在顶层容器内才可以。
- 特殊容器: 在 GUI 上起特殊作用的中间层, 如 JInternalFrame、JLayeredPane、JRootPane。
- 这里的特殊容器类其实也就是中间容器类中的一种, 只不过它在图形上更加能够起到美化和专业化的作用。
- 基本组件: 实现人机交互的组件, 如 JButton、JComboBox、JList、JMenu、JSlider、JTextField。

基本组件是指那些只能依托在中间容器上才能被显示的组件, 它不能独立存在。下面给

一段程序，相信大家一定会明白上面一句话的含义。

```
import javax.swing.*;
public class test
{
    public static void main(String[] args)
    {
        JButton button=new JButton();
    }
}
```

如果运行以上程序的话，运行结果将没有任何显示，因为它必须依托在中间容器中，并让中间容器依托在顶层容器内才会显示出来。将上面的程序修改如下：

```
import javax.swing.*;
public class HelloWorld
{
    public static void main(String[] args)
    {
        JFrame frame=new JFrame("测试窗口");           // 创建顶层容器
        JPanel pane=new JPanel();                       // 创建中间容器
        frame.setContentPane(pane);                     // 将中间容器依托在顶层容器内
        JButton button=new JButton("这是一个测试按钮"); // 创建一个基本组件
        pane.add(button);                               // 将基本组件依托在中间容器内
        frame.setVisible(true);
    }
}
```

上述代码的运行结果如图 3.3 所示。

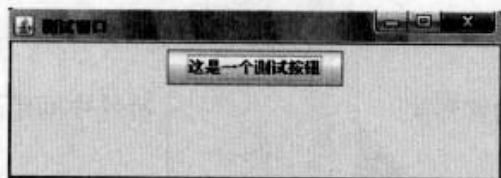


图 3.3 基本组件的显示

此程序可以将基本组件显示出来，从而可以证明基本组件也是不可独立显示的组件。

3.2 Window 类

上一节已经为读者讲述了顶层容器类的基本概念。接下来将会为读者介绍更加深入、更加实用的知识，例如如何使用顶层容器类、顶层容器类的种类，以及顶层容器类所继承下来的层次等，希望读者能够通过仔细的阅读，再加上自己的多多练习来巩固本节知识。

3.2.1 顶层容器类和包含层次

当开发人员使用 Java 进行图形编程的时候，其组件将被绘制在哪里呢？此时，需要一个能够提供图形绘制的容器，这个容器就被称为顶层容器，也可以把它想象成一个窗口。顶层容器是进行图形编程的基础，一切图形化的东西都必然包括在顶层容器中。在 Swing 中主要有三种可以使用的顶层容器：

- JFrame 用来设计类似于 Windows 系统中的窗口形式的应用程序。
- JDialog 和 JFrame 类似，只不过 JDialog 用来设计对话框。
- JApplet 用来设计可以嵌入在网页中的 Java 小程序。

这三种顶层容器都是可以独立显示的。在实际开发中，一般都是将它们作为一个图形界面的最顶层窗口。

基于 Swing 的图形界面至少要有一个顶层容器。容器与其所包含的组件形成了树状包含层次结构，顶层容器就是作为这个包含层次结构的根。每一个顶层容器都有一个内容面板，这个内容面板也就是前面提到的中间容器类组件，该内容面板中可以包含很多界面中所需要的组件。另外，在顶层容器中，也可以添加菜单组件，而菜单组件一般是放在顶层容器中，和内容面板是并行的，也就是说，顶层容器可以同时包含菜单组件和内容面板。其具体的实现过程将在后面为读者详细讲解。

在实际开发中，绝大多数 Java 程序是使用 JFrame 组件对象作为该程序的顶层容器。当然，JDialog 和 JApplet 也可以作为顶层窗口，只不过 JDialog 一般作为弹出窗口来使用，而 JApplet 一般作为嵌在网页中的小程序的框架来使用。大多数的图形界面的顶层容器仍然是使用 JFrame 作为程序框架。

下面给出一个有关 JFrame 作为顶层窗口的示意图，从而使读者能够更加清晰地理解上面所讲述的内容，如图 3.4 所示。

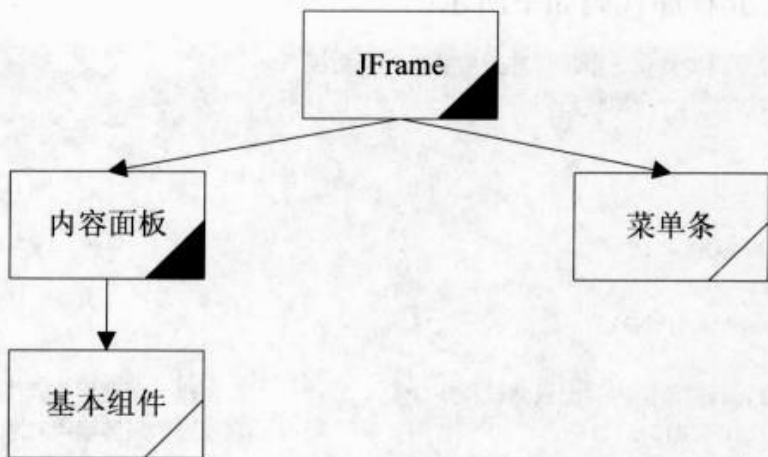


图 3.4 JFrame 作为顶层窗口的层次示意图

在以上的示意图中，可以联系到实际开发中设计一个图形界面的方法，其方法就是首先创建一个顶层容器 JFrame，接下来创建一个内容面板或者中间容器，也可以创建一个菜单组件，最后将所需要的基本组件按照一定的布局方式添加到内容面板中，这样就形成了一个图形界面。

3.2.2 在顶层容器中添加组件

与 AWT 组件不同，Swing 组件不能直接添加到顶层容器中，它必须添加到一个与 Swing 顶层容器相关联的内容面板（ContentPane）上。内容面板其实就是一个中间容器，它是一个轻量级组件。基本规则如下：

- 把 Swing 组件放入一个与顶层 Swing 容器相联系的内容面板上。
- 避免使用非 Swing 的重量级组件。

然而，在 JFrame 顶层容器中添加中间组件和基本组件有两种方式：

- 一种方式是用 getContentPane()方法获得 JFrame 的内容面板，再在这个内容面板中添加组件，方法为 frame.getContentPane().add(childComponent)。
- 另一种方式是首先建立一个 JPanel 或 JDesktopPane 之类的中间容器，把组件添加到容器中，然后再用 setContentPane()方法把该容器置为 JFrame 的内容面板。

下面将通过表格的形式将这两种方法列举出来，如表 3.1 所示。

表 3.1 在顶层容器中添加组件的方法

在顶层容器中添加组件的方法	说明
pane=JFrame.getContentPane(); pane.add(基本组件);	首先使用 pane=JFrame.getContentPane()方法获得内容面板，然后再通过使用 pane.add(基本组件)的方法来添加组件
JPanel pane=new JPanel(); Pane.add(基本组件); JFrame frame=new JFrame(); Frame.setContentPane();	首先使用 JPanel pane=new JPanel()创建一个内容面板，将组件利用 add()方法添加其内，然后再使用 setContentPane()方法把该容器置为 JFrame 的内容面板

根据以上表格所列举出来的方法，下面将通过一个 Swing 程序来为读者讲述如何在顶层容器中添加内容面板。其程序代码如下所示：

```
// 这段代码主要是为读者介绍如何在一个顶层容器内获取一个面板
// 也可以说是在顶层容器内产生一个默认的内容面板
import javax.swing.*;
public class containers
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("添加内容面板测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPanc=new JPanel( );
        jf.setContentPane(contentPanc);
    }
}
```

// 创建一个顶层容器类对象
// 设置顶层容器类对象的大小
// 设置顶层容器类对象的关闭功能
// 设置顶层容器类的可见性
// 创建一个中间容器类对象
// 将中间容器组件对象 contentPanc 设置为内容面板

上面程序的运行结果如图 3.5 所示。



图 3.5 添加内容面板测试程序运行图

上面的程序段主要用于将一个内容面板添加到顶层容器中去，接下来就可以在内容面板中添加其他组件，代码如下所示：

```
// 这段代码主要是为读者介绍如何在已经创建好的内容面板中添加普通组件
import javax.swing.*;
public class containers1
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("添加内容面板测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JButton b1=new JButton("确定");           // 创建两个按钮组件
        JButton b2=new JButton("取消");
        contentPane.add(b1);                       // 在内容面板中添加按钮基本组件
        contentPane.add(b2);                       // 在内容面板中添加按钮基本组件
    }
}
```

上面程序段的运行结果如图 3.6 所示。



图 3.6 在内容面板中添加控件程序运行图

相信通过以上的实例和表格，读者应该能够掌握如何在中间容器中添加组件。下一小节将为读者继续讲述如何在顶层容器中添加菜单栏。

3.2.3 在顶层容器中添加菜单栏

下面将讨论如何在顶层容器中添加菜单栏。同样，将通过编写一段 Swing 程序代码来讲解如何在顶层容器中添加菜单栏。其代码如下：

```
// 这段代码主要是为读者介绍如何将创建好的菜单放置在顶层容器内
import javax.swing.*;
public class menutest
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame();
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



```

jf.setTitle("学生管理系统");
JMenuBar menubar1=new JMenuBar();
jf.setJMenuBar(menubar1);
JMenu menu1=new JMenu("文件");
JMenu menu2=new JMenu("编辑");
JMenu menu3=new JMenu("视图");
menubar1.add(menu1);
menubar1.add(menu2);
menubar1.add(menu3);
JMenuItem item1=new JMenuItem("打开");
JMenuItem item2=new JMenuItem("保存");
JMenuItem item3=new JMenuItem("打印");
JMenuItem item4=new JMenuItem("退出");
menu1.add(item1);
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu1.addSeparator();
menu1.add(item4);
jf.setVisible(true);
}

```

// 添加菜单条组件
// 将菜单条添加到顶层容器中
// 设置菜单组件

// 将菜单组件添加到菜单条组件中

// 创建菜单项组件

// 将菜单项组件添加到相应的菜单组件中去

// 菜单项之间的分隔线组件

上面程序段的运行结果如图 3.7 和图 3.8 所示。



图 3.7 在顶层容器中添加菜单

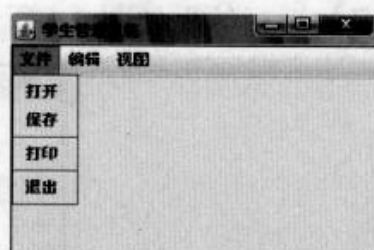


图 3.8 菜单项展开

从上面的程序实例可以看出在顶层容器上是如何添加菜单的。那么在菜单中的空白处是否能够添加基本组件呢？先观察一下如下代码：

```

import javax.swing.*;
public class HelloWorld
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame();
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setTitle("学生管理系统");
        JMenuBar menubar1=new JMenuBar();
        jf.setJMenuBar(menubar1);
        JMenu menu1=new JMenu("文件");
        JMenu menu2=new JMenu("编辑");
        JMenu menu3=new JMenu("视图");
        menubar1.add(menu1);
        menubar1.add(menu2);
    }
}

```

// 添加菜单条组件
// 将菜单条添加到顶层容器中
// 设置菜单组件

// 将菜单组件添加到菜单条组件中



```
menubar1.add(menu3);
menubar1.add(menu4);
JMenuItem item1=new JMenuItem("打开");           // 创建菜单项组件
JMenuItem item2=new JMenuItem("保存");
JMenuItem item3=new JMenuItem("打印");
JMenuItem item4=new JMenuItem("退出");
menu1.add(item1);                                // 将菜单项组件添加到相应的菜单组件中去
menu1.add(item2);
menu1.addSeparator();                            // 菜单项之间的分隔线组件
menu1.add(item3);
menu1.addSeparator();
menu1.add(item4);
jf.setVisible(true);
JButton button=new JButton("这是一个测试按钮组件");
JPanel pane=new JPanel();
pane.add(button);
jf.setContentPane(pane);
}
}
```

上面程序的运行结果如图 3.9 所示。

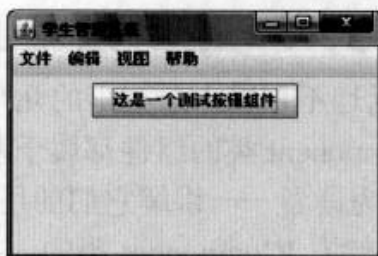


图 3.9 组件添加到菜单的顶层容器中

上面的实例主要是在顶层容器中使用方法 `setJMenuBar()` 嵌入菜单条，再使用方法 `setContentPane()` 嵌入中间容器。按照这样的方式，就可以形成一个完整的应用程序界面。

3.3 JComponent 类

JComponent 类是所有轻量级组件的父类，为了能够让读者更加清晰地了解 JComponent 类的主要子类，这里将以表格的形式列出其子类，如表 3.2 所示。

表 3.2 JComponent 类的子类

子类名称	说明
JButton	创建按钮对象，而且可以创建带图标按钮
JTree	创建树对象
JComboBox	创建组合框对象，和 Choice 相似
JCheckBox	创建复选框对象
JFileChooser	创建文件选择器
JInternalFrame	创建内部窗体
JLabel	创建标签
JMenu	创建菜单对象

子类名称	说明
JMenuBar	创建菜单条对象
JPanel	创建面板对象
JPasswordField	创建口令文本对象
JPopupMenu	创建弹出式菜单
JProgressBar	创建进程条
JScrollBar	创建滚动条
JTextArea	创建文本区
JTable	创建表格
JSplitPane	创建拆分窗格
JToolTip	创建工具提示对象
JToolBar	创建工具条
JTexPane	创建文本窗格
JRadioButton	创建单选按钮
JScrollPane	创建滚动窗格
JSlider	创建滚动条

当然，JComponent 类的子类远远不止以上所列出的几个，这些只是一些常用的子类。从这张表中可以看出，所有继承 JComponent 类的组件都属于基本组件类，它们只能依赖于中间容器而存在。在后面的章节中将会为读者一一讲解它们的用法。

接下来的内容主要为读者讲述有关 JComponent 类的一些特性。那么 JComponent 类有哪些特性呢？其实它的特性非常多，其中有 9 大特性尤其显著，如图 3.10 所示。

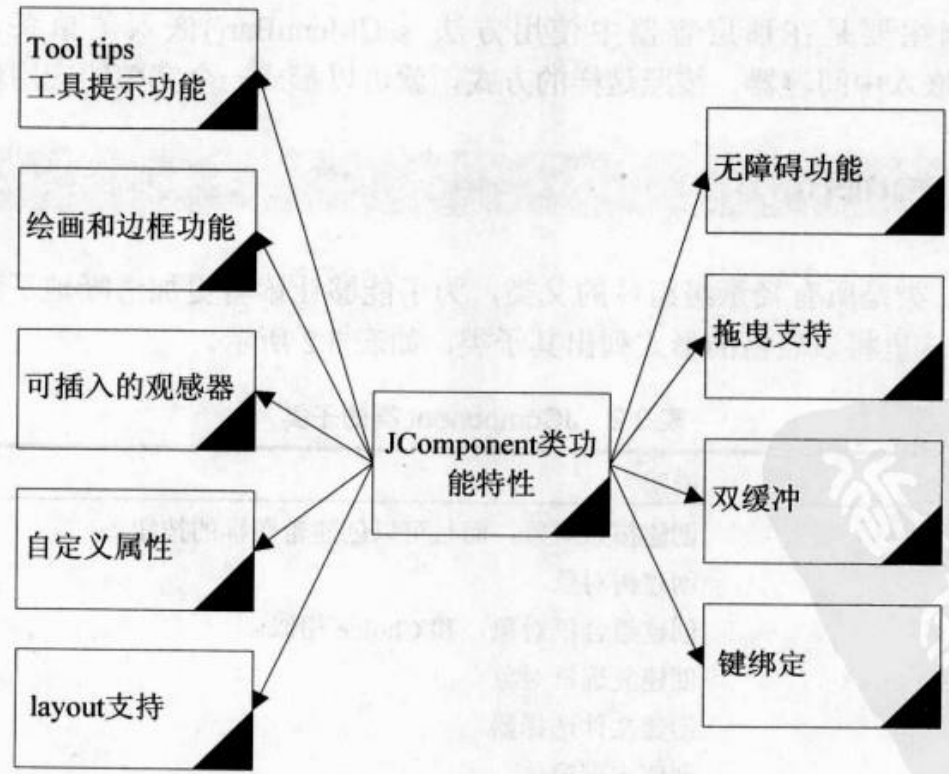


图 3.10 JComponent 类的功能图

下面将讲解这 9 大功能，力求让读者对这些功能有个清晰的认识。

1. Tool tips

Tool tips 就是工具提示功能, 通过在 `setToolTipText` 方法中指定字符串, 可以为一个组件提供一些帮助。当鼠标停留在组件上的时候, 指定的字符串将出现在该组件的附近。下面将通过一个实例来说明它的用法, 其代码如下:

```
// 这段代码主要是使读者能够清楚如何为组件添加工具提示功能
import javax.swing.*;
public class containers2
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("添加内容面板测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        JButton b1=new JButton("确定");
        JButton b2=new JButton("取消");
        contentPane.add(b1);
        contentPane.add(b2);
        b1.setToolTipText("这个按钮是一个确定按钮");// 设置按钮组件的工具提示功能
        b2.setToolTipText("这个按钮是一个取消按钮");
    }
}
```

以上程序段的运行结果如图 3.11 所示。

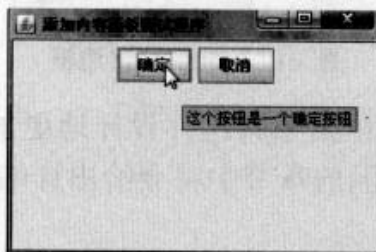


图 3.11 工具提示的程序运行

2. 绘画和边框

在这里将为读者讲述绘画和边框的功能。当一个 Swing 的 GUI 需要绘制自身时, 绘制将从需要绘制的最顶层组件开始, 依据层次关系绘制。这个过程是由 AWT 绘制系统来操作的, 并且通过 Swing 重新绘制管理器等等来最终完成。

每一个 `JComponent` 可以有一个或多个边框。边框是非常有用的对象。虽然边框本身不是组件, 但是它们知道如何绘制 Swing 组件的边界。它们的用途不仅仅局限于绘制线条和漂亮的边界, 还可以提供标题和组件周围的空白控件。

要在一个 `JComponent` 周围设置边框, 可以使用方法 `setBorder`, 也可以使用 `BorderFactory` 类来创建 Swing 所提供的绝大多数的边框。下面将通过一个实例来观察它是如何被实现的, 其代码如下:


```
// 这段代码主要是为读者介绍如何设置按钮组件的边框
import javax.swing.*;
import java.awt.Color;
public class containers3
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("添加内容面板测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JButton b1=new JButton("确定");
        JButton b2=new JButton("取消");
        contentPane.add(b1);
        contentPane.add(b2);
        b1.setBorder(BorderFactory.createLineBorder(Color.red));// 用来设置按钮组件的边框
    }
}
```

上面程序段的运行结果如图 3.12 所示。

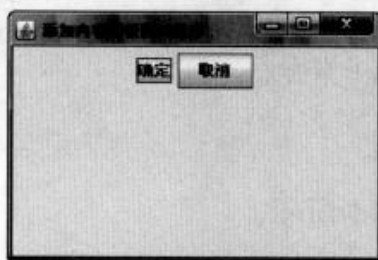


图 3.12 绘制按钮边框

在实际开发中，一般会使用边框对象将组件设计得更加美观，从而满足用户对软件界面的需求。针对这个框架问题，在后面的章节中将会给出详细的分析与讲解。

3. 可插入的观感器

所谓的对可插入观感的支持，也就是可以定制自己的桌面、更换新的颜色方案，让窗口系统适应用户的习惯和需要。这种体系结构使得界面可以显示出不同的风格。Swing 提供了一些早已被打包成形的观感，包括默认、Motif 和 Windows 的 L&F。

当一个程序没有设置外观感觉时，系统必须确定使用哪种外观感觉。它首先将会检查程序代码中是否指定了一个外观感觉。如果已经指定了，它将使用程序员所指定的外观感觉；如果没有指定，系统将选择 Java 外观感觉。

4. 自定义属性

JComponent 的自定义属性可以将一个或者多个属性与任何 JComponent 相关联。例如：一个布局管理器可能使用属性将一个约束对象与它管理的每个 JComponent 相关联。使用 putClientProperty 方法和 getClientProperty 方法可以设置和获得属性值。在现实的开发过程中，会经常使用到这个特性。

5. layout 支持

layout 支持也就是对布局的支持功能。虽然 Component 类提供了布局的方法, 例如 `getPreferredSize` 方法和 `getAlignmentX` 方法, 但是如果不创建子类并重载一些方法, 它将不能提供任何方法来设置这些布局。要通过某种方式来设置布局, JComponent 类添加了一些设置方法, 例如: `setPreferredSize`、`setMinimumSize`、`setMaximumSize`、`setAlignmentX` 和 `setAlignmentY` 等。通过设置组件最大(最小)推荐尺寸的方法和设置 X(Y) 对齐参数值的方法能指定布局管理器的约束条件, 从而为布局提供支持。

6. 无障碍

什么是无障碍技术呢? 其实无障碍技术就是允许残疾人士使用计算机。如果用户患有腕管综合症, 可以在不使用手的情况下通过无障碍技术来完成工作。

无障碍技术一般由语音接口、屏幕阅读器、其他的输入设备等组成, 不仅对残疾人士, 而且对在非办公室环境使用计算机的人来说都是非常有用的, 例如当堵车时, 可以使用无障碍技术, 通过语音输入和输出来查收电子邮件。

7. 拖曳支持

在程序启动时, 组件并没有开启对拖动功能的支持, 但在程序窗口的底部有一个复选框允许启用拖动功能。

8. 双缓冲

使用双缓冲技术能改进频繁变化的组件的显示效果。与 AWT 组件不同, JComponent 组件默认双缓冲区, 不必自己重写代码。如果想关闭双缓冲区, 可以在组件上应用 `setDoubleBuffered(false)` 方法。

9. 键绑定

这个特性使组件能够在用户按下键盘特定键后对用户操作进行响应, 例如在许多外观感觉中, 当一个按钮拥有焦点时, 按下空格键就相当于用户用鼠标单击了该按钮。外观感觉自动将按下空格键和释放的操作与单击按钮和释放的操作效果相绑定, 这样的话, 用户既可以通过鼠标来操作, 也可以通过键盘的快捷键来操作, 从而方便了用户对软件的操作。

3.4 本章小结

本章主要讲述了 Swing 组件的框架, 通过对本章的学习, 读者可以清楚如何在顶层容器中放置内容面板、如何在内容面板里放置组件, 以及在顶层容器中放置菜单栏。并且通过最后一节的讲述, 使读者能够对 JComponent 类的特性有进一步地了解。随着学习地深入, 读者会渐渐体会到这些特性。

从下一章开始, 将为读者逐一介绍 Swing 组件类中各种各样的组件的使用方法。

3.5 本章习题

1. 编写一个实例，其中有 4 个按钮组件和一个标签组件，如何将其添加到一个自定义的顶层容器中？

要求：编写的代码的最后运行结果如图 3.13 所示。

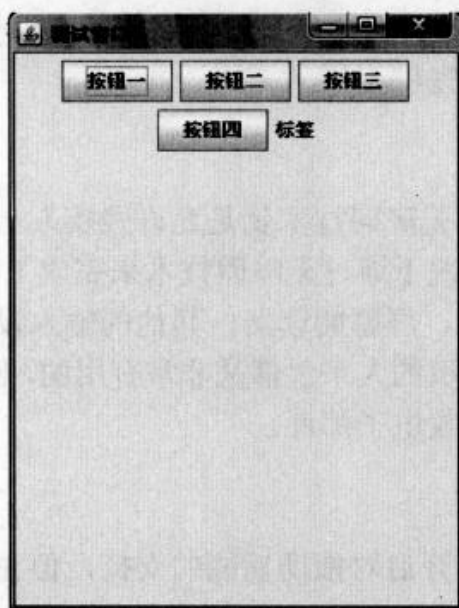


图 3.13 本章习题 1

2. 编写一个实例，其中有一个菜单组件和 4 个标签组件，如何将菜单组件和 4 个标签组件添加到自定义的顶层容器中？

要求：编写的代码的最后运行结果如图 3.14 所示。

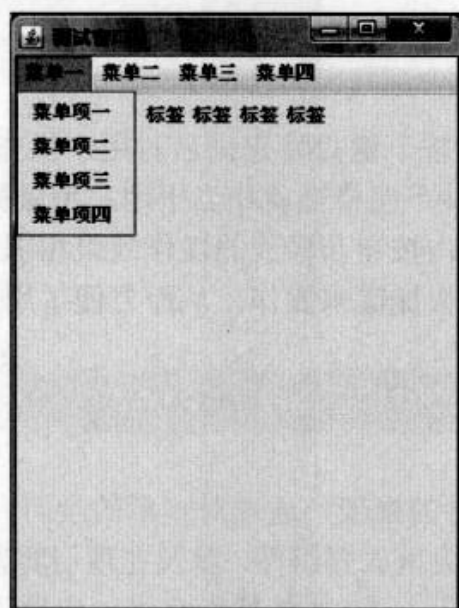


图 3.14 本章习题 2

3. 编写一个实例，在一个顶层容器中添加三个文本组件，这三个文本组件形成了一个乘法表。

要求:

(1) 编写的代码运行后如图 3.15 所示。

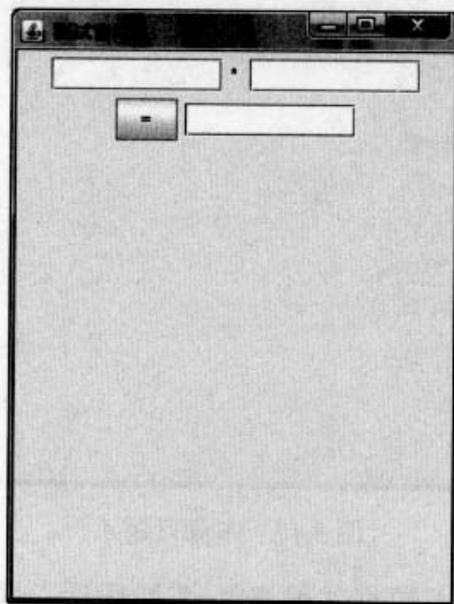


图 3.15 本章习题 3 (a)

(2) 在前面两个文本框中输入数字, 当单击“=”按钮后, 其乘法的结果会显示在第三个文本框中, 如图 3.16 所示。

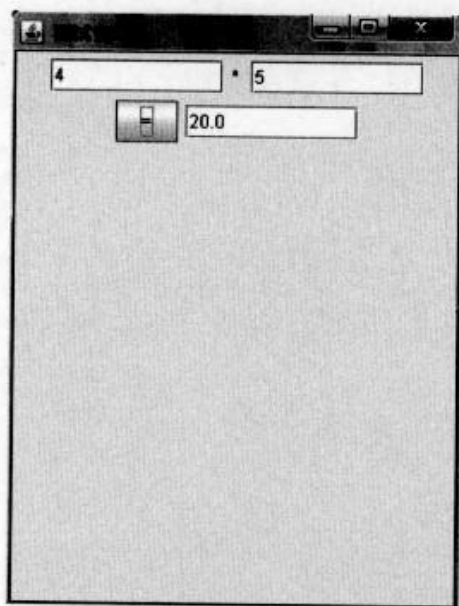


图 3.16 本章习题 3 (b)

4. 编写一个实例, 在一个顶层容器中添加两个文本组件, 使得它们可以形成一个正切函数。

要求:

(1) 其实现的效果如图 3.17 所示。

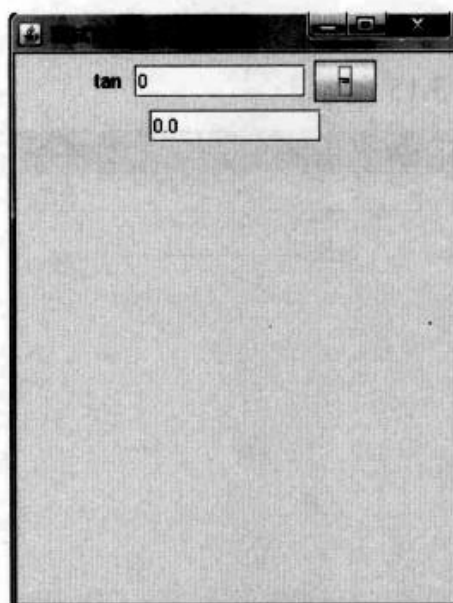


图 3.17 本章习题 4

(2) 当在第一个文本框中输入数据并单击“=”按钮后，就会在第二个文本框中出现结果。



第4章 如何使用标签和按钮组件

标签和按钮是开发一个图形界面必须使用到的基本组件之一。这些组件虽然很简单，但也是最常用的，几乎所有的程序界面都会用到它们。标签用于标识名称、说明性文字。一般来说，标签所显示的文本是不会变的，也是禁止编辑的，然而，也可以通过代码的方式让标签的文字发生改变。按钮也是一个很常见的组件，它是实现人机对话最基本的组件，例如在 ATM 取款机上，当输入密码后就会有几个按钮控件以供用户选择。在本章中将会通过大量的程序实例为读者讲述它们。

4.1 如何使用标签

本节将讲述如何使用标签组件，在使用标签组件之前，首先必须要创建标签组件对象，而后才可以通过使用这个对象内置的方法来操纵它。一般来说，组件都是通过其自身的构造器来创建的，所以下面将先介绍标签的构造器的种类，表 4.1 中列出了 Label 的常用构造器。

表 4.1 Label 常用构造器

Label 常用构造器	说明
Public JLabel()	创建无图像并且其标题为空字符串的 JLabel 对象
Public JLabel(String text)	使用指定的字符串 text（也就是标签显示的文本）创建一个标签
Public JLabel(String text,int alignment)	使用指定的字符串 text 创建一个标签。参数 alignment 用于指定标签文本的对齐方式：左对齐（LEFT）、右对齐（RIGHT）和居中（CENTER）
Public JLabel (Icon image)	创建具有指定图像的 JLabel 对象
Public JLabel(Icon image,int horizontalAlignment)	创建具有指定图像和水平对齐方式的 JLabel 对象
Public JLabel (String text, Icon icon, int horizontalAlignment)	创建具有指定文本、图像和水平对齐方式的 JLabel 对象

通过构造器创建了 Label 组件后，可通过组件内置的方法来操纵这个组件，其方法如表 4.2 所示。

表 4.2 Label 常用方法

Label 常用方法	说明
Public String getText()	获取标签的文本
Public void setText(String text)	设定标签文本

为了使读者熟悉这个组件的用法，接下来将给出一个程序实例，在顶层容器中添加两个标签组件，其具体的代码如下：

```
// 这段代码主要是为读者展示了创建标签的方法，并且通过在构造器中为标签初始赋值
import javax.swing.*;
public class test1
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JLabel label1=new JLabel("这是一个标签测试程序");           // 创建两个标签组件
        JLabel label2=new JLabel("这是一个不可编辑的标签控件");
        contentPane.add(label1);                                     // 将标签组件添加到内容面板中
        contentPane.add(label2);
    }
}
```

以上代码的运行结果如图 4.1 所示。

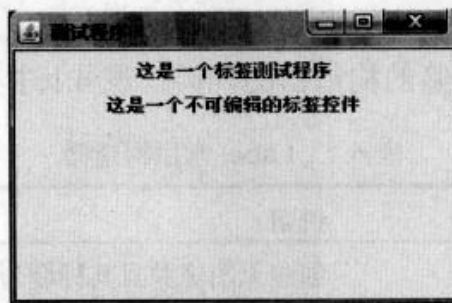


图 4.1 使用对象初始化的方式赋值

以上是使用标签对象初始化的方法为标签赋初始值，下面再使用 JLabel 的方法来给标签赋值。在这个程序中将通过赋值的形式创建标签，下面将给出其程序片断。

```
// 这段代码主要是为读者展示使用 setText()的方法为标签初始赋值
import javax.swing.*;
public class test2
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JLabel label1=new JLabel();
```



```

JLabel label2=new JLabel();
label1.setText("标签是用来标识某个控件的控件");
label2.setText("标签是用来标识说明性文件的控件");
contentPane.add(label1);
contentPane.add(label2);
    }
}

```

以上程序代码的运行结果如图 4.2 所示。

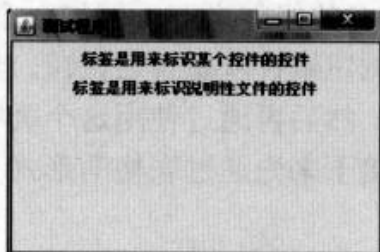


图 4.2 使用标签方法赋值

以上的程序实例充分说明了一个 JLabel 组件的创建和使用方法, 其实大多数组件的创建、使用都是如此, 当学习到后面的组件时, 相信读者会有此体会。

4.2 如何使用按钮

按钮组件是一个非常重要的组件之一, 它也是一个能够实现人机交互操作的典型组件之一。JButton 的层次结构如图 4.3 所示。

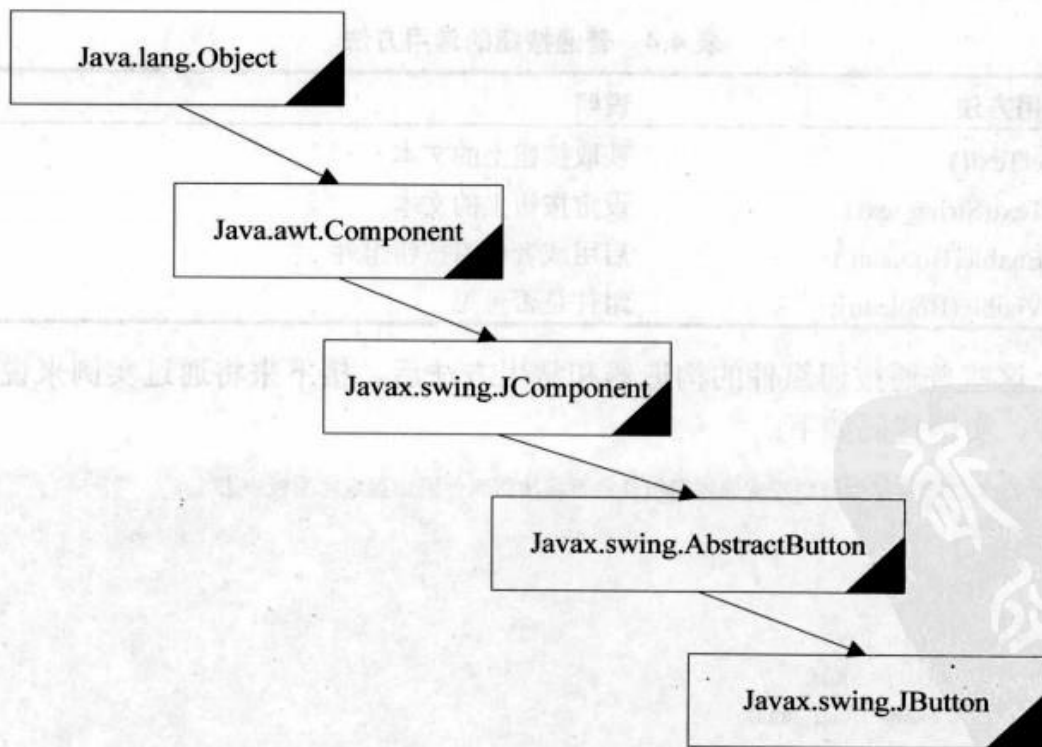


图 4.3 JButton 的层次结构示意图

由以上的 JButton 的层次结构图可以看出, JButton 是继承 AbstractButton 类而来。那什么是 AbstractButton 类呢? 其实, AbstractButton 本身是一个抽象类, 里面定义了许多组件设置的方法与组件事件驱动方法 (Event Handle), 如 addActionListener()、setText()等, 所提供的方

法不少于 50 多种，是非常重要的一个类。事实上，AbstractButton 类不但被 JButton 继承，同时还被 JMenuItem、JToggleButton、JCheckBox、JRadioButton 等类继承。

按钮组件有很多种，其中包括普通按钮、单选按钮、复选框。下面将通过实例的方式为读者讲解每种按钮控件的创建和使用方法。

4.2.1 如何使用普通按钮

普通按钮是平时生活中遇到最多的组件之一，例如一个登录窗口，要求输入用户名和密码后，让用户选择确定还是取消的按钮组件就是普通按钮。当然，与标签组件一样，要使用它，首先必须创建一个按钮组件类对象，然后再通过使用这个类中内置的方法来操纵该对象。为了能够让读者理解起来更加系统化，接下来先通过表格的形式介绍普通按钮组件的构造器，如表 4.3 所示。

表 4.3 普通按钮的常用构造器

普通按钮常用构造器	说明
JButton()	创建不带设置文本或图标的按钮
JButton(Icon icon)	创建一个带图标的按钮
JButton(String text)	创建一个带文本的按钮
JButton(String text,Icon icon)	创建一个带初始文本和图标的按钮

创建好按钮组件后，接下来就可以通过使用其内置的方法来操纵这个组件了，其常用方法如表 4.4 所示。

表 4.4 普通按钮的常用方法

普通按钮的常用方法	说明
Public String getText()	获取按钮上的文本
Public void setText(String text)	设定按钮上的文本
Public void setEnable(Boolean)	启用或者禁用按钮组件
Public void setVisible(Boolean);	组件是否可见

熟悉以上这些普通按钮组件的构造器和常用方法后，接下来将通过实例来说明如何使用普通按钮组件。实例代码如下：

```
// 这段代码主要为读者展示如何创建普通按钮组件，再将按钮组件添加到内容面板中去
import javax.swing.*;
public class test3
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
    }
}
```



```
JButton b1=new JButton("确定");           // 创建两个按钮，并且将按钮添加到内容面板中
JButton b2=new JButton("取消");
contentPane.add(b1);
contentPane.add(b2);
}
}
```

以上程序代码的运行结果如图 4.4 所示。



图 4.4 普通按钮控件

普通按钮控件的使用很简单，而在实际的开发中最关键的就是对按钮组件进行事件侦听。随着学习地深入，大家会慢慢体会到，组件使用最关键的是如何编写控件事件侦听。

4.2.2 如何使用单选按钮

单选按钮也被称为 RadioButton，它通过 JRadioButton 类实现，例如在一些管理软件中会出现“性别”单选按钮，通过选择不同的单选按钮来实现不同性别的选择。单选按钮如图 4.5 所示。



图 4.5 单选按钮示例图

要使用单选按钮，首先必须创建它，而后再通过其内置方法来操纵组件，所以下面将以表格的形式给出单选按钮的构造器，如表 4.5 所示。

表 4.5 单选按钮的构造器

单选按钮构造器	说明
JRadioButton()	创建一个初始化为未选择的单选按钮，其文本未设定
JRadioButton(Icon icon)	创建一个初始化为未选择的单选按钮，其具有指定的图像但无文本
JRadioButton(Icon icon,Boolean selected)	创建一个具有指定图像和选择状态的单选按钮，但无文本
JRadioButton(String text)	创建一个具有指定文本的状态为未选择的单选按钮
JradioButton(String text,Boolean selected)	创建一个具有指定文本和选择状态的单选按钮
JRadioButton(String text,Icon icon)	创建一个具有指定的文本和图像并初始化为未选择的单选按钮
JradioButton(String text,Icon icon, Boolean selected)	创建一个具有指定的文本、图像和选择状态的单选按钮

上面列出了有关单选按钮组件的构造器，下面依旧为读者讲述其内置的常用方法。单选按钮的常用方法同普通按钮的常用方法差不多，这里不再列出。下面将通过实例来观察如何使用单选按钮。其代码如下所示：


```
// 这段代码主要是为读者展示如何创建单选按钮组件
import javax.swing.*;
public class test4
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JRadioButton jr1=new JRadioButton("忽略");    // 创建三个单选按钮，并且将它们添加到内容面板中
        JRadioButton jr2=new JRadioButton("继续");
        JRadioButton jr3=new JRadioButton("跳过");
        contentPane.add(jr1);
        contentPane.add(jr2);
        contentPane.add(jr3);
    }
}
```

上面程序代码的运行结果如图 4.6 所示。

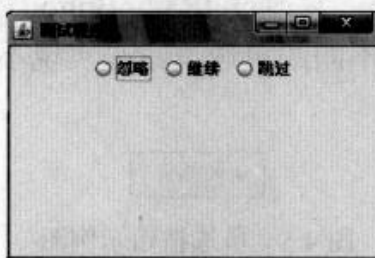


图 4.6 如何添加单选按钮

在以上代码中，3 个单选按钮同时都可以被选中，这个就不符合实际开发中所用到的单选按钮的要求。在实际开发中，一个单选按钮被选中，其他的自动被置为未选中状态，要产生这种效果，就要使用到按钮组。下面将通过实例来说明如何将几个单选按钮划分到按钮组中。其代码如下所示：

```
// 这段代码主要是为读者展示如何创建单选按钮组件
// 并且将这些单选按钮组件添加到一个按钮组中
import javax.swing.*;
public class test5
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
    }
}
```



```

JRadioButton jr1=new JRadioButton("忽略");           // 创建三个单选按钮
JRadioButton jr2=new JRadioButton("继续");
JRadioButton jr3=new JRadioButton("跳过");
ButtonGroup bg=new ButtonGroup();                     // 将三个单选按钮划分到一个按钮组中
bg.add(jr1);
bg.add(jr2);
bg.add(jr3);
contentPane.add(jr1);                                // 将三个单选按钮添加到内容面板中
contentPane.add(jr2);
contentPane.add(jr3);
}

```

上面程序代码的运行结果如图 4.7 所示。

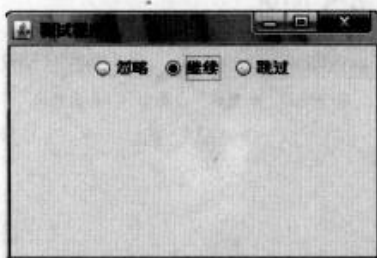


图 4.7 将单选按钮划分到按钮组中

经过上述程序代码的处理后，图中的三个按钮只能在同一个时间内选择一个，而不能同时选择多个。另外，有一个实际应用的小技巧，当一个界面上有很多个单选按钮时，可以将同一类的单选按钮划分到同一个按钮组中，这样就可以形成多个按钮组，用于执行特殊的功能。下面将给出一个综合实例，这个实例将多个单选按钮组件分别分成多个按钮组，其具体代码如下所示：

```

// 这段代码主要是为读者展示如何创建单选按钮组件，以及如何将它们放在不同的按钮组中
import javax.swing.*;
public class test6
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JRadioButton jr1=new JRadioButton("羽毛球");
        // 创建 6 个单选按钮，并且将之分成三组按钮组
        JRadioButton jr2=new JRadioButton("足球");
        JRadioButton jr3=new JRadioButton("电脑书");
        JRadioButton jr4=new JRadioButton("数学书");
        JRadioButton jr5=new JRadioButton("电影");
        JRadioButton jr6=new JRadioButton("录像");
        ButtonGroup bg1=new ButtonGroup();
        ButtonGroup bg2=new ButtonGroup();
        ButtonGroup bg3=new ButtonGroup();
    }
}

```



```
        bg1.add(jr1);
        bg1.add(jr2);
        bg2.add(jr3);
        bg2.add(jr4);
        bg3.add(jr5);
        bg3.add(jr6);
        contentPane.add(jr1);      // 将 6 个单选按钮添加到内容面板中
        contentPane.add(jr2);
        contentPane.add(jr3);
        contentPane.add(jr4);
        contentPane.add(jr5);
    }
}
```

上面程序代码的运行结果如图 4.8 所示。

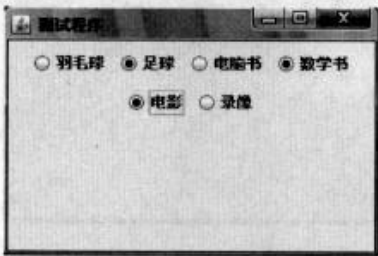


图 4.8 多个按钮组

在上面的实例中，如果选中了“羽毛球”单选按钮就不能选择同类的“足球”，同样如果选择了“电脑书”单选按钮，就不能选择同类的“数学书”单选按钮。

4.2.3 如何使用复选框

复选框利用 JCheckbox 类实现，它跟单选按钮的区别就是复选框可以多选，单选按钮只能单选。仍然像介绍其他组件一样，先以表格的形式为大家介绍复选框的构造器，如表 4.6 所示。

表 4.6 复选框的构造器

复选框构造器	说明
JCheckbox()	创建一个初始化为未选择的复选框，其文本未设定
JCheckbox (Icon icon)	创建一个初始化为未选择的复选框，其具有指定的图像但无文本
Jcheckbox (Icon icon,Boolean selected)	创建一个具有指定图像和选择状态的复选框，但无文本
JCheckbox (String text)	创建一个具有指定文本的状态为未选择的复选框
Jcheckbox (String text,Boolean selected)	创建一个具有指定文本和选择状态的复选框
JCheckbox (String text,Icon icon)	创建一个具有指定文本和图像并初始化为未选择的复选框
JCheckbox (String text,Icon icon,Boolean selected)	创建一个具有指定文本、图像和选择状态的复选框

下面将通过实例来说明如何使用复选框。其代码如下所示：

```
// 这段代码主要是为读者展示如何创建复选框组件
import javax.swing.*;
public class test7
{
    static final int WIDTH=300;
```



```

static final int HEIGHT=200;
public static void main(String[] args)
{
    JFrame jf=new JFrame("测试程序");
    jf.setSize(WIDTH,HEIGHT);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setVisible(true);
    JPanel contentPane=new JPanel( );
    jf.setContentPane(contentPane);
    JCheckBox jc1=new JCheckBox("羽毛球");
    // 创建 6 个复选框，并且将之添加到内容面板中
    JCheckBox jc2=new JCheckBox("足球");
    JCheckBox jc3=new JCheckBox("电脑书");
    JCheckBox jc4=new JCheckBox("数学书");
    JCheckBox jc5=new JCheckBox("电影");
    JCheckBox jc6=new JCheckBox("录像");
    contentPane.add(jc1);
    contentPane.add(jc2);
    contentPane.add(jc3);
    contentPane.add(jc4);
    contentPane.add(jc5);
    contentPane.add(jc6);
}
}

```

以上程序代码的运行结果如图 4.9 所示。

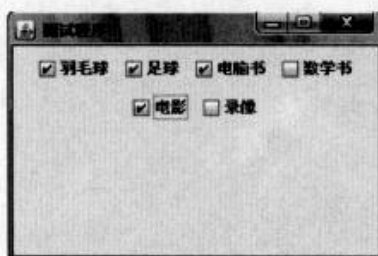


图 4.9 如何添加复选框

复选框比较适合多重选择的情况，但是它与单选按钮的多重选择又不一样。它是没有单选限制的，而前面讲的单选按钮的多重选择是有限制的。请读者仔细体会两者的不同。

4.2.4 按钮组件的实例应用

本小节将针对前面讨论过的几个不同的按钮组件给予综合运用。下面将通过一个实例将所有类的按钮组件一一绘制在顶层容器内。实例代码如下所示：

```

// 这段代码用于复习前面所讲过的普通按钮组件、单选按钮组件、复选框组件、标签组件
// 利用这些组件创建一个信息窗口，其中包括姓名、兴趣爱好、性别等信息
import javax.swing.ButtonGroup;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
public class test8 extends JPanel
{

```



```
static final int WIDTH=300;
static final int HEIGHT=200;
test8()
{
    JFrame frame=new JFrame();
    frame.setTitle("如何使用按钮的测试程序");
    frame.setSize(WIDTH,HEIGHT);
    frame.setContentPane(this);
    JLabel name=new JLabel("王鹏");
    JRadioButton jr1=new JRadioButton("男");
    JRadioButton jr2=new JRadioButton("女");
    ButtonGroup bg=new ButtonGroup();
    bg.add(jr1);           // 将两个单选按钮划分到同一个单选按钮组 bg 中
    bg.add(jr2);
    JLabel interesting=new JLabel("兴趣爱好");
    JCheckBox jc1=new JCheckBox("羽毛球");
    JCheckBox jc2=new JCheckBox("足球");
    JCheckBox jc3=new JCheckBox("电脑书");
    JCheckBox jc4=new JCheckBox("数学书");
    JCheckBox jc5=new JCheckBox("电影");
    JCheckBox jc6=new JCheckBox("录像");
    add(name);
    add(jr1);
    add(jr2);
    add(interesting);
    add(jc1);
    add(jc2);
    add(jc3);
    add(jc4);
    add(jc5);
    add(jc6);
    frame.setVisible(true);
}
public static void main(String[] args)
{
    new test();
}
```

上面程序代码的运行结果如图 4.10 所示。

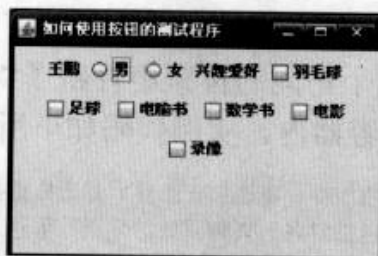


图 4.10 综合运用

上面的程序实例中包含了本章所讲述的所有控件，但是不知道读者是否发现整个窗口里的控件排列不是很令人满意，这是因为布局的问题，后面会将布局管理单独列出来进行详解，这里不再赘述。

下面将针对本章所讲述的内容做一个总结。

- 在顶层窗口中添加标签: 只须将标签控件添加到顶层窗口的内容面板上即可。至于标签上的文字, 可以使用构造器和 `setText()` 两种方法来创建。
- 普通按钮控件的添加: 只须将按钮控件添加到顶层窗口的内容面板上即可。至于按钮上的文字, 可以使用构造器来创建。
- 单选按钮控件的添加: 只须将按钮控件添加到顶层窗口的内容面板上即可。至于按钮旁边的文字, 可以使用构造器来创建。另外如果要想实现单选的目的, 就需要将单选的一批按钮划分到按钮组中。
- 复选按钮控件的添加: 只须将按钮控件添加到顶层窗口的内容面板上即可。至于按钮旁边的文字, 可以使用构造器来创建。

4.3 本章小结

本章通过大量的实例向读者介绍了标签和按钮组件的使用, 并总结了使用这些组件的经验。

在下一章中将针对本章遗留下来的布局管理方面的知识给予详细讨论。

4.4 本章习题

1. 编写一个实例, 将 3 个按钮、3 个标签、2 个文本组件添加到自定义的顶层容器中, 当在第 1 个文本框中输入数据并单击按钮时, 在第 2 个文本框中会显示出结果。

要求:

(1) 这个题目的最终效果如图 4.11 所示。

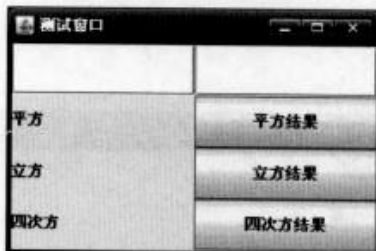


图 4.11 本章习题 1 (a)

(2) 当单击按钮组件时, 3 个标签组件的文本将会发生变化, 其效果如图 4.12 所示。

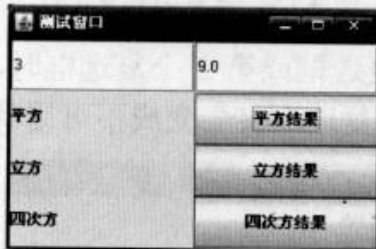


图 4.12 本章习题 1 (b)

这个题目涉及到按钮动作事件, 至于如何使用, 在后面的章节中将会给出详细讲述, 这里不过多讲解。

2. 编写一个实例, 共 6 个单选按钮, 分为两组, 当单击第 1 组的第 1 个单选按钮或第 2 个单选按钮时, 另一个就会呈现不可选状态, 当单击第 3~6 个单选按钮时, 其余 3 个单选按钮将会变成不可用状态。

要求:

(1) 此实例的最终效果如图 4.13 所示。

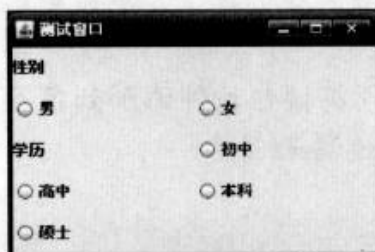


图 4.13 本章习题 2 (a)

(2) 当单击第 1 组的第 1 个单选按钮或第 2 个单选按钮时, 另一个就会呈现不可选状态, 当单击第 3~6 个单选按钮时, 其余 3 个单选按钮将会变成不可用状态, 其效果如图 4.14 所示。

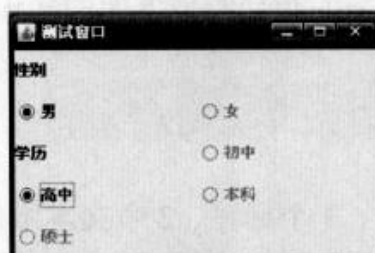


图 4.14 本章习题 2 (b)

3. 将上面实例中的单选按钮修改成复选框。

要求:

(1) 最终效果如图 4.15 所示。

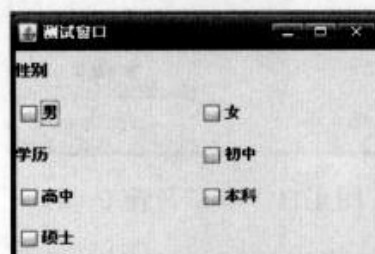


图 4.15 本章习题 3 (a)

(2) 当单击第 1 组的第 1 个复选框或第 2 个复选框时, 另一个就会呈现不可选状态, 当单击第 3~6 个复选框时, 其余 3 个复选框将会变成不可用状态, 其效果如图 4.16 所示。

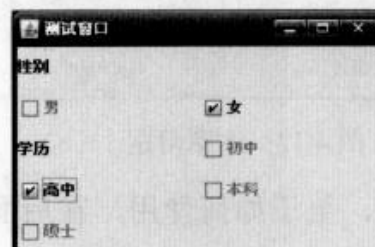


图 4.16 本章习题 3 (b)

4. 将上例进行修改, 添加两个普通按钮组件, 当单击普通按钮时, 会起到与上例相同的作用。

要求:

(1) 最终效果如图 4.17 所示。

图 4.17 本章习题 4 (a)

(2) 当单击第 1 个复选框或第 2 个复选框时, 再单击普通按钮, 另一个就会呈现不可用状态, 当单击第 3~6 个复选框, 再单击普通按钮, 其余 3 个复选框将会变成不可用状态, 如图 4.18 所示。

图 4.18 本章习题 4 (b)

第5章 如何使用布局管理器组件

在上一章的最后一个实例中，还遗留一个问题，即顶层窗口控件排列的不是很令人满意，其原因就是没有对顶层容器进行布局管理。本章将针对布局管理器进行详细讨论。所谓的布局管理器，就是为容器内的组件提供了若干布局策略，每个容器都拥有某种默认布局管理器，用于负责其内部组件的排列。在目前的开发中，常用的布局管理器有 BorderLayout、FlowLayout、GridLayout、GridBagLayout、CardLayout、BoxLayout、SpringLayout、GroupLayout 等。下面将针对以上几种布局管理器，通过大量的实例为读者进行详细分析、讲解和总结。

5.1 布局管理器概述

Java 中的布局管理功能，没有 Visual Basic、PowerBuilder 等可视化编程软件所带的表单设计器，可以在画板上直接将组件按照程序员的意愿进行排列。在 Java 中只能通过编写代码来规范每个控件在顶层窗口中的位置。不同的代码代表着不同的布局管理方式，不同的布局管理方式对应着不同风格的布局格调。在设计软件的时候，必须要根据用户的要求，选择不同风格的布局管理器。为了能够让读者清楚每个布局管理器的意义，下面将以表格的形式列出，如表 5.1 所示。

表 5.1 布局管理器的种类

种类	说明
BorderLayout	它将容器分为 5 个部分，即东、南、西、北、中，每一个区域中可以容纳一个组件，使用的时候也是通过 BorderLayout 中的 5 个方位常量来确定组件所在的位置
FlowLayout	是按加入的先后顺序从左到右排列，一行排满了，再换行，继续从左到右排列。每一行的组件都是居中排列的
GridLayout	是将整个布局空间，划分成若干行乘若干列的网络区域。组件就位于这些小的区域内
GridBagLayout	是通过网格进行划分，可以看到每个组都占据一个网格，也可以一个组件占据几个网格。有点类似于 GirdLayout，但是比它复杂的多
CardLayout	将容器中的每一个组件当作一个卡片，一次仅有一个卡片可见，最初显示容器时，增加到 CardLayout 对象的第一个组件可见
BoxLayout	通过允许在容器中水平或垂直的方式安排多个组件
SpringLayout	通过定义组件边沿的关系来实现布局
GroupLayout	指定在一个窗体上组件彼此之间的关系，例如一个位置关系或对齐关系

以上的所有布局管理器各有各的特色，它们适合于不同的场合。

布局管理器能够将各种组件在顶层容器内排列的井井有条，那么，布局管理器为什么能够有如此能力？它是如何工作的呢？下面将讲述有关布局管理器的工作原理。

在创建好顶层框架后，会调用 `JFrame` 的 `pack` 方法，用于指定顶层框架所必须的首选大小。而这个首选大小是框架的内容窗口大小与框架菜单栏的大小之和。内容窗口的布局管理器主要负责计算内容窗口的首选大小，例如要使用一个具有两列的 `GridLayout` 来布局，那么系统会将所有的组件的大小都设置为一样的，并且使得每个组件的高度和宽度与所有组件中高度和宽度最大的组件相同。通过这种方式计算出内容窗口的首选大小。然后，根据每个组件的大小，按照先后或者坐标位置，将之放入到布局管理器所布局的组件中去。

5.2 布局管理器的种类

和 AWT 相同，为了容器中的组件能实现与平台无关的自动合理排列，Swing 也采用了布局管理器来管理组件的排放、位置、大小等布置任务，在此基础上将显示风格做了改进。另外一个不同点在于，Swing 虽然有顶层容器，但是不能把组件直接加到顶层容器中，Swing 窗体中含有一个称为内容面板的容器（`ContentPane`），也可以说是中间容器。在顶层容器上放置内容面板，然后把组件加入到内容面板中，所以在 Swing 中，设置布局管理器是针对内容面板的，另外 Swing 新增加了一个 `BoxLayout` 布局管理器，其显示与 AWT 略有不同。

5.2.1 BorderLayout

`BorderLayout` 是一种简单的布局策略，在使用这个布局管理器的时候，应该将其看作是一个组件，所以，首先应通过构造器创建布局管理器对象，再通过引用其中的方法和变量来对组件进行布局。下面将以表格的形式列举出 `BorderLayout` 布局管理器的构造器，如表 5.2 所示。

表 5.2 BorderLayout 布局管理器的构造器

布局管理器的构造器	说明
<code>BorderLayout()</code>	构造一个组件之间没有间距的新边框布局
<code>BorderLayout(int h,int v)</code>	构造一个具有指定组件间距的边框布局

这个布局管理器把容器分为东、南、西、北、中 5 个区域，每个组件将占据某个区域。而这 5 个区域分别被命名为 `NORTH`、`WEST`、`EAST`、`CENTER`、`SOUTH`，它们都被定义为静态常量，静态常量可以直接引用。这种布局管理器所定义的 5 个静态常量的说明如表 5.3 所示。

表 5.3 BorderLayout 布局管理器的静态常量表

常量	说明
<code>Public static final String NORTH="North"</code>	整个内容面板的北边
<code>Public static final String WEST="West"</code>	整个内容面板的西边
<code>Public static final String EAST="East"</code>	整个内容面板的东边
<code>Public static final String CENTER="Center"</code>	整个内容面板的中间
<code>Public static final String SOUTH="South"</code>	整个内容面板的南边

当向某个区域内添加控件时,就要将代表区域的常数作为第2个参数传递给add方法函数,而将需要添加到某个区域的控件作为 add 方法的第 1 个参数,如 add(组件名称,方位)。下面将通过一个实例为读者展示 BorderLayout 布局管理器的使用方法,其代码如下所示:

```
// 这段代码主要是为读者展示如何使用 BorderLayout 布局管理器将组件进行布局
import javax.swing.*;
import java.awt.*;
public class test1
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        JButton b1=new JButton("生活");
        JButton b2=new JButton("工作");
        JButton b3=new JButton("睡觉");
        JButton b4=new JButton("购物");
        JButton b5=new JButton("饮食");
        BorderLayout lay=new BorderLayout();
        // 创建一个布局管理器对象, 将中间容器设置为此布局管理
        jf.setLayout(lay);
        contentPane.add(b1,"North");
        // 将 5 个普通按钮组件分别按照东、南、西、北、中 5 个方位添加到中间容器中
        contentPane.add(b2,"South");
        contentPane.add(b3,"East");
        contentPane.add(b4,"West");
        contentPane.add(b5,"Center");
    }
}
```

上面程序代码的运行结果如图 5.1 所示。

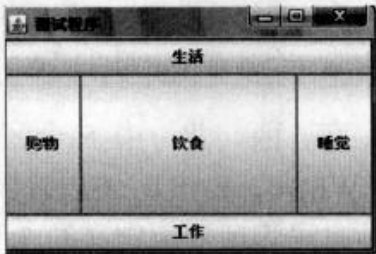


图 5.1 BorderLayout 布局管理器

这种布局管理器将 5 个控件分别放在 5 个不同的位置上,为了能够更加熟悉这种布局管理器,下面将列举一个稍微复杂的实例。在实例中将 5 个不同的内容面板放到与顶层窗口相关联的内容面板后,再在每个内容面板中添加组件,其代码如下所示:

```
// 这段代码展示了如何将 5 个内容面板添加到与顶层容器相关联的内容面板中的 5 个不同方向
// 这 5 个内容面板中也分别包含了位于 5 个不同方向的按钮组件
import javax.swing.*;
import java.awt.*;
public class test2
```



```
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        // 创建了 25 个普通按钮组件
        JButton b1=new JButton("港币");
        JButton b2=new JButton("人民币");
        JButton b3=new JButton("美元");
        JButton b4=new JButton("欧元");
        JButton b5=new JButton("英镑");
        JButton b6=new JButton("主板");
        JButton b7=new JButton("内存");
        JButton b8=new JButton("硬盘");
        JButton b9=new JButton("显示器");
        JButton b10=new JButton("鼠标");
        JButton b11=new JButton("大米");
        JButton b12=new JButton("蔬菜");
        JButton b13=new JButton("稻子");
        JButton b14=new JButton("猪肉");
        JButton b15=new JButton("牛肉");
        JButton b16=new JButton("面包");
        JButton b17=new JButton("蛋糕");
        JButton b18=new JButton("巧克力");
        JButton b19=new JButton("奶酪");
        JButton b20=new JButton("苹果派");
        JButton b21=new JButton("笔记本");
        JButton b22=new JButton("电话");
        JButton b23=new JButton("办公桌");
        JButton b24=new JButton("钢笔");
        JButton b25=new JButton("文件夹");
        jf.setLayout(new BorderLayout());
        JPanel p1=new JPanel();
        // 创建了 5 个中间容器, 并且将它们布局管理器设置成 BorderLayout 方式
        JPanel p2=new JPanel();
        JPanel p3=new JPanel();
        JPanel p4=new JPanel();
        JPanel p5=new JPanel();
        p1.setLayout(new BorderLayout());
        p2.setLayout(new BorderLayout());
        p3.setLayout(new BorderLayout());
        p4.setLayout(new BorderLayout());
        p5.setLayout(new BorderLayout());
        contentPane.add(p1,"North");
        // 将 5 个中间容器对象分别加入到上层中间容器中, 并且按照 BorderLayout 的方式进行布局
        contentPane.add(p2,"South");
        contentPane.add(p3,"East");
        contentPane.add(p4,"West");
        contentPane.add(p5,"Center");
        // 将第 1~5 个普通按钮组件按照 BorderLayout 方式布局到 p1 中间容器中
        p1.add(b1,"North");
        p1.add(b2,"West");
    }
}
```



```

p1.add(b3,"South");
p1.add(b4,"East");
p1.add(b5,"Center");
// 将第 6~10 个普通按钮组件按照 BorderLayout 方式布局到 p2 中间容器中
p2.add(b6,"North");
p2.add(b7,"West");
p2.add(b8,"South");
p2.add(b9,"East");
p2.add(b10,"Center");
// 将第 11~15 个普通按钮组件按照 BorderLayout 方式布局到 p3 中间容器中
p3.add(b11,"North");
p3.add(b12,"West");
p3.add(b13,"South");
p3.add(b14,"East");
p3.add(b15,"Center");
// 将第 16~20 个普通按钮组件按照 BorderLayout 方式布局到 p4 中间容器中
p4.add(b16,"North");
p4.add(b17,"West");
p4.add(b18,"South");
p4.add(b19,"East");
p4.add(b20,"Center");
// 将第 21~25 个普通按钮组件按照 BorderLayout 方式布局到 p5 中间容器中
p5.add(b21,"North");
p5.add(b22,"West");
p5.add(b23,"South");
p5.add(b24,"East");
p5.add(b25,"Center");
}

```

上面的程序代码的运行结果如图 5.2 所示。

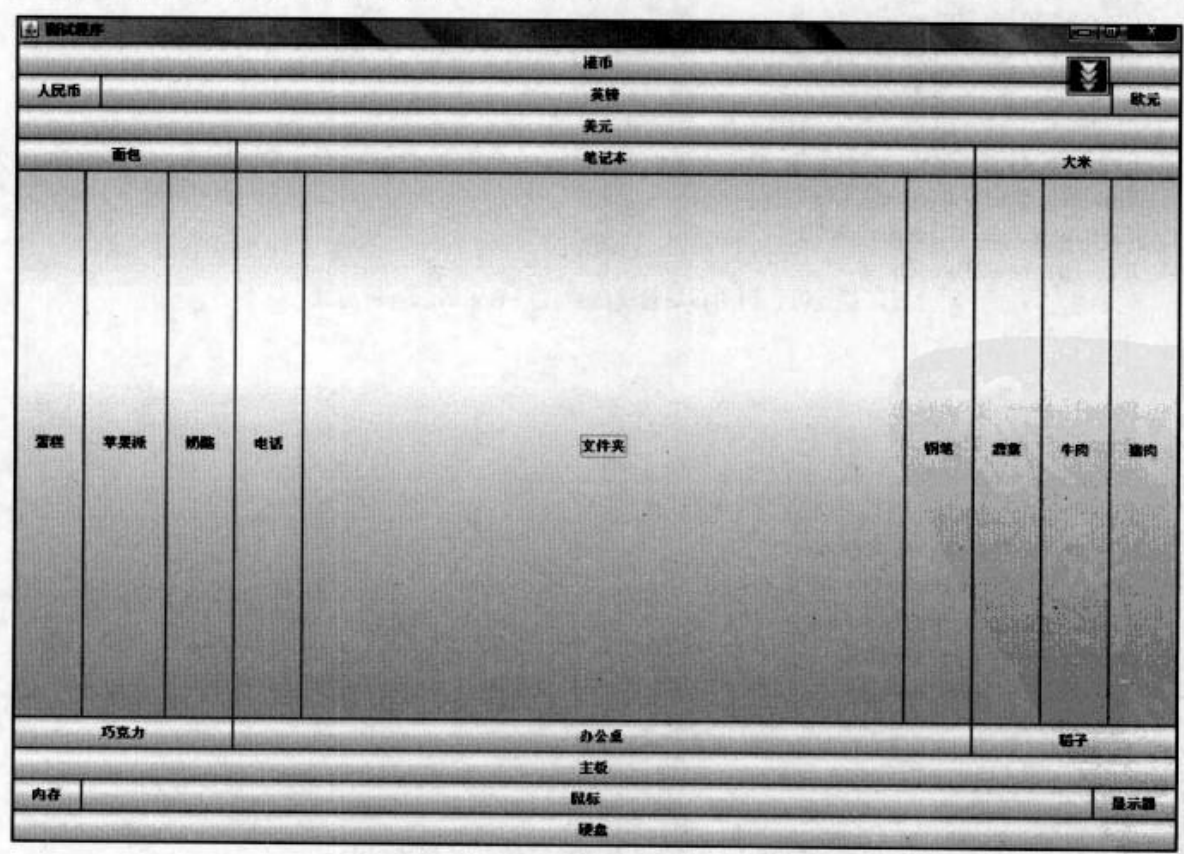


图 5.2 复杂的 BorderLayout 布局管理器

上述实例使用了布局管理器内嵌布局管理器的方式。每个方位都添加一个中间容器，每个中间容器又使用一个 BorderLayout 布局管理器来布局其中的组件。当然，这种界面并不令人满意。随着学习地深入，后面会介绍更加个性化、人性化的布局管理器，它们将会使开发人员开发出专业、华丽、实用的图形界面，从而能够最大程度地满足用户对软件界面的需求。

5.2.2 FlowLayout

这种布局管理器的策略也非常简单，它是按照控件加入的先后顺序从左到右排列，一行排满了，再换下一行，然后继续从左到右排列。每一行的组件都是居中排列的。另外如果有些按钮看不到，可以使用方法 pack 自动调整 Frame 的大小，使得所有控件都显示出来。FlowLayout 布局管理器同样是通过先创建对象、再利用其内置方法和变量来布局的组件，如表 5.4 所示为其构造器的说明。

表 5.4 FlowLayout 布局管理器的构造器

构造器	说明
FlowLayout()	构造一个 FlowLayout 对象，它是居中对齐的，默认的水平 and 垂直间隙是 5 个单位
FlowLayout(int align)	构造一个 FlowLayout 对象，默认的水平 and 垂直间隙是 5 个单位
FlowLayout(int align,int h,int v)	创建一个新的流布局管理器，它具有指定的对齐方式以及指定的水平和垂直间隙

学习了 FlowLayout 布局管理器的构造器后，下面将通过实例来讲述此布局管理器的使用方法。其代码如下所示：

```
// 这段代码主要是为读者展示 FlowLayout 布局管理器的使用方法
import javax.swing.*;
import java.awt.*;
public class test3
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        JButton b1=new JButton("港币");
        JButton b2=new JButton("人民币");
        JButton b3=new JButton("美元");
        JButton b4=new JButton("欧元");
        JButton b5=new JButton("英镑");
        contentPane.setLayout(new FlowLayout()); // 将中间容器的布局管理器设置为 FlowLayout
        contentPane.add(b1); // 将 5 个按钮分别按照 FlowLayout 布局管理器方式添加到中间容器中
        contentPane.add(b2);
        contentPane.add(b3);
        contentPane.add(b4);
    }
}
```



```
contentPane.add(b5);
jf.pack();
}
```

上面程序代码的运行结果如图 5.3 所示。

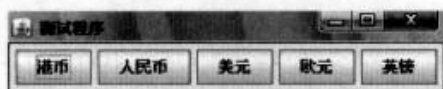


图 5.3 FlowLayout 布局管理器

由图 5.3 可知，所有组件都是依次排列的，没有像 BorderLayout 布局管理器一样有 5 个不同的方位。以上的实例比较简单，下面将 BorderLayout 布局管理器和 FlowLayout 布局管理器结合在一起列举一个实例。此实例的思路是将 FlowLayout 布局管理器同顶层容器关联，然后在其中添加 5 个 BorderLayout 布局管理器，并在这 5 个布局管理器中添加组件。其代码如下所示：

```
// 为每一个内容面板添加 5 个组件，并按照 BorderLayout 布局管理方式排列组件
import javax.swing.*;
import java.awt.*;
public class test4
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        // 创建了 25 个普通按钮组件
        JButton b1=new JButton("港币");
        JButton b2=new JButton("人民币");
        JButton b3=new JButton("美元");
        JButton b4=new JButton("欧元");
        JButton b5=new JButton("英镑");
        JButton b6=new JButton("主板");
        JButton b7=new JButton("内存");
        JButton b8=new JButton("硬盘");
        JButton b9=new JButton("显示器");
        JButton b10=new JButton("鼠标");
        JButton b11=new JButton("大米");
        JButton b12=new JButton("蔬菜");
        JButton b13=new JButton("稻子");
        JButton b14=new JButton("猪肉");
        JButton b15=new JButton("牛肉");
        JButton b16=new JButton("面包");
        JButton b17=new JButton("蛋糕");
        JButton b18=new JButton("巧克力");
        JButton b19=new JButton("奶酪");
        JButton b20=new JButton("苹果派");
```



```

JButton b21=new JButton("笔记本");
JButton b22=new JButton("电话");
JButton b23=new JButton("办公桌");
JButton b24=new JButton("钢笔");
JButton b25=new JButton("文件夹");
contentpane.setLayout(new FlowLayout());
// 将中间容器的布局管理器设为 FlowLayout
JPanel p1=new JPanel();      // 创建 5 个中间容器, 并且将每个中间容器的布局管理器设置为 BorderLayout
JPanel p2=new JPanel();
JPanel p3=new JPanel();
JPanel p4=new JPanel();
JPanel p5=new JPanel();
p1.setLayout(new BorderLayout());
p2.setLayout(new BorderLayout());
p3.setLayout(new BorderLayout());
p4.setLayout(new BorderLayout());
p5.setLayout(new BorderLayout());
contentPane.add(p1);        // 将 5 个中间容器添加到上层中间容器
contentPane.add(p2);
contentPane.add(p3);
contentPane.add(p4);
contentPane.add(p5);
// 将第 1~5 个普通按钮添加到 p1 中
p1.add(b1,"North");
p1.add(b2,"West");
p1.add(b3,"South");
p1.add(b4,"East");
p1.add(b5,"Center");
p2.add(b6,"North");        // 将第 6~10 个普通按钮添加到 p2 中
p2.add(b7,"West");
p2.add(b8,"South");
p2.add(b9,"East");
p2.add(b10,"Center");
p3.add(b11,"North");       // 将第 10~15 个普通按钮添加到 p3 中
p3.add(b12,"West");
p3.add(b13,"South");
p3.add(b14,"East");
p3.add(b15,"Center");
p4.add(b16,"North");       // 将第 16~20 个普通按钮添加到 p4 中
p4.add(b17,"West");
p4.add(b18,"South");
p4.add(b19,"East");
p4.add(b20,"Center");
p5.add(b21,"North");       // 将第 21~25 个普通按钮添加到 p5 中
p5.add(b22,"West");
p5.add(b23,"South");
p5.add(b24,"East");
p5.add(b25,"Center");
jf.pack();
}
}

```

上面程序代码的运行结果如图 5.4 所示。



图 5.4 BorderLayout 和 FlowLayout 布局管理器相结合

以上实例给出了两个不同布局管理器在同一个内容面板中的不同表现形式。读者可以尝试着将两个布局管理器对调一下，也就是使用 BorderLayout 布局管理器与顶层窗口关联，使用 FlowLayout 布局管理器来布局控件，读者可以看看结果如何。

5.2.3 GridLayout

这种布局管理器有点像围棋盘，它将整个布局空间划分成若干行乘若干列的网络区域。组件就位于这些小的区域内，要想创建一个 GridLayout 布局管理器，就必须通过其构造器来创建 GridLayout 布局管理器对象。下面将通过表格的形式给出 GridLayout 布局管理器的构造器，如表 5.5 所示。

表 5.5 GridLayout 布局管理器的构造器

GridLayout 布局管理器的构造器	说明
GridLayout()	构造一个组件之间没有间距的新边框布局
GridLayout(int h,int v)	构造一个具有指定组件间距的边框布局，h、v 是指行数和列数

下面将通过实例来熟悉 GirdLayout 布局管理器的使用方法，其代码如下所示：

```
// 这段代码主要是为读者展示如何使用 GridLayout 布局管理器
// 在程序中将 9 个普通按钮组件按照此布局管理器放置在内容面板中
import javax.swing.*;
import java.awt.*;
public class test5
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        JButton b1=new JButton("港币");
        JButton b2=new JButton("人民币");
        JButton b3=new JButton("美元");
        JButton b4=new JButton("欧元");
        JButton b5=new JButton("英镑");
        JButton b6=new JButton("主板");
        JButton b7=new JButton("内存");
        JButton b8=new JButton("硬盘");
        JButton b9=new JButton("显示器");
        GridLayout gird=new GridLayout(3,3);
```



```
// 创建一个 GridLayout 布局管理器对象
// 将之行数设为 3，列数设为 3，并且将之作为中间容器的布局管理器
contentPane.setLayout(gird);
contentPane.add(b1);
// 将 9 个普通按钮组件——添加到中间容器中
contentPane.add(b2);
contentPane.add(b3);
contentPane.add(b4);
contentPane.add(b5);
contentPane.add(b6);
contentPane.add(b7);
contentPane.add(b8);
contentPane.add(b9);
jf.pack();
}
```

上面程序代码的运行结果如图 5.5 所示。



图 5.5 GridLayout 布局管理器

从上图中可以看出，9 个普通按钮组件是按照三行、三列的方式进行排列的。下面将 GridLayout、BorderLayout、FlowLayout 布局管理器结合起来设计一个综合实例，主要思路是用 GridLayout 布局管理器与顶层窗口关联，在这个布局管理器中添加 BorderLayout 布局管理器和 BorderLayout 布局管理器，最后在这些布局管理器中添加控件。实例代码如下所示：

```
// 这段代码主要是用 GridLayout 布局管理器与顶层窗口关联
// 在这个布局管理器中添加 BorderLayout 布局管理器和 BorderLayout 布局管理器
// 最后，在这些布局管理器中添加控件
import javax.swing.*;
import java.awt.*;
public class test6
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        // 创建了 25 个普通按钮组件
        JButton b1=new JButton("港币");
        JButton b2=new JButton("人民币");
        JButton b3=new JButton("美元");
        JButton b4=new JButton("欧元");
        JButton b5=new JButton("英镑");
        JButton b6=new JButton("主板");
```



```

JButton b7=new JButton("内存");
JButton b8=new JButton("硬盘");
JButton b9=new JButton("显示器");
JButton b10=new JButton("鼠标");
JButton b11=new JButton("大米");
JButton b12=new JButton("蔬菜");
JButton b13=new JButton("稻子");
JButton b14=new JButton("猪肉");
JButton b15=new JButton("牛肉");
JButton b16=new JButton("面包");
JButton b17=new JButton("蛋糕");
JButton b18=new JButton("巧克力");
JButton b19=new JButton("奶酪");
JButton b20=new JButton("苹果派");
JButton b21=new JButton("笔记本");
JButton b22=new JButton("电话");
JButton b23=new JButton("办公桌");
JButton b24=new JButton("钢笔");
JButton b25=new JButton("文件夹");
GridLayout gird=new GridLayout(3,3);
jf.setLayout(gird);
// 创建 5 个中间容器, 并且将第 1 个、第 2 个、第 4 个、第 5 个中间容器的布局方式设置为 BorderLayout
// 而第 3 个设置为 FlowLayout
JPanel p1=new JPanel();
JPanel p2=new JPanel();
JPanel p3=new JPanel();
JPanel p4=new JPanel();
JPanel p5=new JPanel();
p1.setLayout(new BorderLayout());
p2.setLayout(new BorderLayout());
p3.setLayout(new FlowLayout());
p4.setLayout(new BorderLayout());
p5.setLayout(new BorderLayout());
contentPane.add(p1);           // 将 5 个中间容器添加到外层中间容器中
contentPane.add(p2);
contentPane.add(p3);
contentPane.add(p4);
contentPane.add(p5);
p1.add(b1,"North");           // 将第 1~5 个组件添加到 p1
p1.add(b2,"West");
p1.add(b3,"South");
p1.add(b4,"East");
p1.add(b5,"Center");
p2.add(b6,"North");           // 将第 6~10 个组件添加到 p2
p2.add(b7,"West");
p2.add(b8,"South");
p2.add(b9,"East");
p2.add(b10,"Center");
p3.add(b11);                  // 将第 11~15 个组件添加到 p3
p3.add(b12);
p3.add(b13);
p3.add(b14);
p3.add(b15);
p4.add(b16,"North");          // 将第 16~20 个组件添加到 p4
p4.add(b17,"West");
```



```

p4.add(b18,"South");
p4.add(b19,"East");
p4.add(b20,"Center");
p5.add(b21,"North");           // 创建第 21~25 个组件添加到 p5
p5.add(b22,"West");
p5.add(b23,"South");
p5.add(b24,"East");
p5.add(b25,"Center");
jf.pack();
}

```

上面程序代码的运行结果如图 5.6 所示。



图 5.6 BorderLayout、FlowLayout、GridLayout 布局管理器相结合

上面的实例是将前面介绍的三种布局管理器结合起来使用，这样做的目的在于让读者能够更加熟悉、灵活地运用各种布局管理器。

5.2.4 GridBagLayout

GridBagLayout 是一种很先进的、很人性化的布局管理器，通过网格的划分，可以看到每个组件都占据一个网格，也可以一个组件占据几个网格。与 GridLayout 布局管理器不同的是，GridBagLayout 是按照开发人员自己的思路来排列控件位置，而 GridLayout 布局管理器根据系统的安排来布局。如果要采用网格组布局管理器，一般来说可以采用下列步骤：

- 01 创建一个 GridBagLayout 对象。
- 02 将容器设成此对象的布局管理器。
- 03 创建约束 (GridBagConstraints) 对象。
- 04 创建各个相应的组件。
- 05 添加各个组件与约束到网格组布局管理器中。

网格组由多个网格组成，而且各个行或者列的长度和宽度不同。但默认情况下，单元格从左上角开始有序列的编号，从第 0 行、第 0 列开始计数。

当向网格组布局管理器中添加组件时，需要分别定义每个单元格的序列号，只要设定相应的值，那么组件就会添加到网格组布局管理器中。涉及到组件被添加到什么位置有 4 个参数，即 gridX、gridY、gridwidth、gridheight。其中，gridX、gridY 分别定义了添加组件时左上角的行与列的位置，而 gridwidth、gridheight 分别定义了组件所占用的列数和行数。

网格组布局管理器中每个区域都要设置增量字段 (weightx 与 weighty 分别代表 x 方向和 y

这两个参数都是非常重要的约束，其中当组件不能填满单元格时，fill 参数就可以发挥作用。该约束的值主要有以下几种。

- 而 `anchor` 参数则是当一个组件大于分配给它的单元格时发挥作用，该约束就是约定如何处理该组件，它的值如下所示。

- GridBagLayout 布局管理器的构造器只有一种，就是不带参数的构造器，这里不再列举。以上简单讲述了 GridBagLayout 布局管理器的一些理论知识，下面将通过实例让读者熟悉它的使用方法，实例代码如下所示：

```
Dimension screenSize=kit.getScreenSize();
```



```

int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
loginframe.setLocation(x,y);
JButton ok=new JButton("确认");
JButton cancel=new JButton("放弃");
JLabel title=new JLabel("布局管理器测试窗口");
JLabel name=new JLabel("用户名");
JLabel password=new JLabel("密 码");
final JTextField nameinput=new JTextField(15);
final JTextField passwordinput=new JTextField(15);
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.anchor=GridBagConstraints.EAST;
constraints.weightx=3;
constraints.weighty=4;
add(title,constraints,0,0,4,1);
add(name,constraints,0,1,1,1);
add(password,constraints,0,2,1,1);
add(nameinput,constraints,2,1,1,1);
add(passwordinput,constraints,2,2,1,1);
add(ok,constraints,0,3,1,1);
add(cancel,constraints,2,3,1,1);
loginframe.setResizable(false);
loginframe.setVisible(true);
}
}
public class test7
{
    public static void main(String[] args)
    {
        login log=new login();
    }
}

```

// 使用网格组布局添加控件

上面程序代码的运行结果如图 5.7 所示。

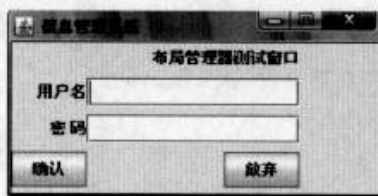


图 5.7 GridBagLayout 布局管理器

5.2.5 CardLayout

CardLayout 布局管理器非常简单，它将容器中的每一个组件当作一个卡片，一次仅有一个卡片可见，如最初显示容器时，CardLayout 对象的第一个组件可见，其他的组件都是不可见的。下面将以表格的形式给出 CardLayout 布局管理器的构造器，如表 5.6 所示。

表 5.6 CardLayout 布局管理器的构造器

CardLayout 布局管理器的构造器	说明
public CardLayout()	创建一个间距大小为 0 的新卡片布局
public CardLayout(int hgap,int vgap)	创建一个新卡片布局管理器，其中 hgap 和 vgap 分别为卡片间水平和垂直方向上的空白空间

在这个布局管理器中有一些常用方法，其方法说明如表 5.7 所示。

表 5.7 CardLayout 布局管理器的常用方法

CardLayout 布局管理器的常用方法	说明
public void first(Container parent)	移到指定容器的第一个卡片
public void next(Container parent)	移到指定容器的下一个卡片
public void previous(Container parent)	移到指定容器的前一个卡片
public void last(Container parent)	移到指定容器的最后一个卡片
public void show(Container parent,String name)	显示指定卡片

以上介绍了 CardLayout 的构造器和常用方法，下面设计一个 CardLayout 程序实例，代码如下：

```
// 这段程序代码用于展示如何使用 CardLayout 布局管理器针对内容面板中的组件进行布局
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.UIManager;
public class test8 extends JFrame
{
    // 主要的 JPanel，该 JPanel 的布局管理将被设置成 CardLayout
    private JPanel pane = null;
    private JPanel p = null;                                // 放按钮的 JPanel
    private CardLayout card = null;                          // CardLayout 布局管理器
    private JButton button_1 = null;                         // 上一步
    private JButton button_2 = null;                         // 下一步
    private JButton b_1 = null, b_2 = null, b_3 = null;      // 三个可直接翻转到 JPanel 组件的按钮
    private JPanel p_1 = null, p_2 = null, p_3 = null;      // 要切换的三个 JPanel
    public test8()
    {
        super("CardLayout Test");
        try
        {
            // 将 LookAndFeel 设置成 Windows 样式
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.
```



```
WindowsLookAndFeel");
}
catch (Exception ex)
{
    ex.printStackTrace();
}
card = new CardLayout(5, 5);
pane = new JPanel(card);
p = new JPanel();
button_1 = new JButton("< 上一步");
button_2 = new JButton("下一步 >");
b_1 = new JButton("1");
b_2 = new JButton("2");
b_3 = new JButton("3");
b_1.setMargin(new Insets(2,2,2,2));
b_2.setMargin(new Insets(2,2,2,2));
b_3.setMargin(new Insets(2,2,2,2));
p.add(button_1);
p.add(b_1);
p.add(b_2);
p.add(b_3);
p.add(button_2);
p_1 = new JPanel();
p_2 = new JPanel();
p_3 = new JPanel();
p_1.setBackground(Color.RED);
p_2.setBackground(Color.BLUE);
p_3.setBackground(Color.GREEN);
p_1.add(new JLabel("JPanel_1"));
p_2.add(new JLabel("JPanel_2"));
p_3.add(new JLabel("JPanel_3"));
pane.add(p_1, "p1");
pane.add(p_2, "p2");
pane.add(p_3, "p3");
// 下面是翻转到卡片布局的某个组件的动作事件处理
// 当单击某个普通按钮组件时, 就会触发出现下一个组件
button_1.addActionListener(new ActionListener()
{
    // 上一步的按钮动作
    public void actionPerformed(ActionEvent e)
    {
        card.previous(pane);
    }
});
button_2.addActionListener(new ActionListener()
{
    // 下一步的按钮动作
    public void actionPerformed(ActionEvent e)
    {
        card.next(pane);
    }
});
b_1.addActionListener(new ActionListener()
{
    // 直接翻转到 p_1
    public void actionPerformed(ActionEvent e)
```

// 创建一个具有指定水平和垂直间隙的新卡片布局
// JPanel 的布局管理将被设置成 CardLayout
// 构造放按钮的 JPanel


```
{ card.show(pane, "p1");
}
});
b_2.addActionListener(new ActionListener()
{
    // 直接翻转到 p_2
    public void actionPerformed(ActionEvent e)
    {
        card.show(pane, "p2");
    }
});
b_3.addActionListener(new ActionListener()
{
    // 直接翻转到 p_3
    public void actionPerformed(ActionEvent e)
    {
        card.show(pane, "p3");
    }
});
this.getContentPane().add(pane);
this.getContentPane().add(p, BorderLayout.SOUTH);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setSize(300, 200);
this.setVisible(true);
}
public static void main(String[] args)
{
    new test8();
}
}
```

上面程序代码的运行结果如图 5.8 所示。



图 5.8 CardLayout 实例 (a)

当单击 2 按钮后，界面如图 5.9 所示。



图 5.9 CardLayout 实例 (b)

当单击“下一步”按钮后，界面如图 5.10 所示。



图 5.10 CardLayout 实例 (c)

在上述实例中，当单击任何一个按钮组件时，框中的、埋藏在下面的标签就会显示出来。这种布局管理器相对简单，以上的程序涉及到了动作事件处理方面的知识，后面将进行详细讲解，这里不再赘述。

5.2.6 BorderLayout

箱式布局比较灵活，也比较实用。Swing 提供的 BOX 类就是箱式布局类，它的默认布局管理器就是 BorderLayout，在箱式布局管理器中包括两种箱子：一种是水平箱，另外一种垂直箱。

创建一个水平箱的源代码如下：

```
Box horBox=Box.createHorizontalBox();
```

创建一个垂直箱的源代码如下：

```
Box verBox=Box.createVerticalBox();
```

创建好箱子后，就可以像添加其他组件一样添加下面的控件，源代码如下：

```
horBox.add(okButton);
verBox.add(cancelButton);
```

两种箱子的区别在于组件的排列顺序，水平箱是按照从左到右的顺序排列，而垂直箱按照从上到下的顺序排列。对于箱式布局管理器而言，最关键的就是每个组件的 3 个尺寸。

- 首选尺寸，即组件显示时的宽度和高度。
- 最大尺寸，即组件能被显示的最大宽度和高度。
- 最小尺寸，即组件被显示的最小高度和最小宽度。

下面是水平箱式布局管理器中组件排列的几个重点：

- 计算最高组件的最大高度，尝试把所有的组件都增加到这个高度。如果有某些组件不能达到此高度，那么要在 Y 轴上对齐需要通过 `getAlignmentY` 方法实现，该方法返回一个介于 0（按顶部对齐）和 1（按底部对齐）之间的浮点数。组件的默认值是 0.5，也就是中线对齐。
- 得到每个组件的首选宽度，然后把所有的首选宽度合计起来。
- 如果首选宽度总和小于箱的宽度，那么所有的组件都会相应的延伸，直到适应这个箱子的宽度。组件从左到右排列，并且相邻两个组件之间没有多余的空格。

箱式布局组件之间没有空隙，那么就要通过一个称为填充物的组件来提供空隙。箱式布局管理器提供了 3 种填充物：支柱、固定区、弹簧。下面将通过实例来介绍如何使用箱式布局管理器布局。实例代码如下所示：

```
// 这段程序主要是为读者展示如何使用 BorderLayout 布局管理器针对组件进行布局
import javax.swing.*;
import java.awt.*;
public class test9
{
    public static void main(String[] args)
    {
        BorderLayoutFrame frame1=new BorderLayoutFrame();
        frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame1.show();
    }
}
class BorderLayoutFrame extends JFrame
{
    private static final int WIDTH=300;
    private static final int HEIGHT=200;
    public BorderLayoutFrame()
    {
        setTitle("测试箱式布局管理器");           // 设置顶层容器名称、大小
        setSize(WIDTH,HEIGHT);
        Container con=getContentPane();           // 创建一个中间容器
        JLabel label1=new JLabel(" 姓名: ");       // 创建标签组件、文本框组件
        JTextField textField1=new JTextField(10);
        textField1.setMaximumSize(textField1.getPreferredSize());
        Box hbox1=Box.createHorizontalBox();       // 创建一个水平箱子
        hbox1.add(label1);
        // 在水平箱子上添加一个标签组件，并且创建一个不可见的 20 个单位的组件
        // 在这之后再添加一个文本框组件
        hbox1.add(Box.createHorizontalStrut(20));
        hbox1.add(textField1);
        JLabel label2=new JLabel(" 密码: ");       // 创建标签组件、文本框组件
        JTextField textField2=new JTextField(10);
        textField2.setMaximumSize(textField2.getPreferredSize());
        Box hbox2=Box.createHorizontalBox();       // 创建一个水平箱子
        hbox2.add(label2);
        // 在水平箱子上添加一个标签组件，并且创建一个不可见的 20 个单位的组件
        // 在这之后再添加一个文本框组件
        hbox2.add(Box.createHorizontalStrut(20));
        hbox2.add(textField2);
        JButton button1=new JButton("确定");
        // 创建两个普通按钮组件，并且创建一个水平箱子，将两个按钮添加到箱子中
        JButton button2=new JButton("取消");
        Box hbox3=Box.createHorizontalBox();
        hbox3.add(button1);
        hbox3.add(button2);
        Box vbox=Box.createVerticalBox();
        // 创建一个垂直箱子，这个箱子将两个水平箱子添加到其中，创建一个横向 glue 组件
        vbox.add(hbox1);
        vbox.add(hbox2);
        vbox.add(Box.createVerticalGlue());
    }
}
```



```

vbox.add(hbox3);
// 将垂直箱子添加到 BorderLayout 布局管理器中的中间位置
con.add(vbox, BorderLayout.CENTER);
}
}

```

上面程序代码的运行结果如图 5.11 所示。

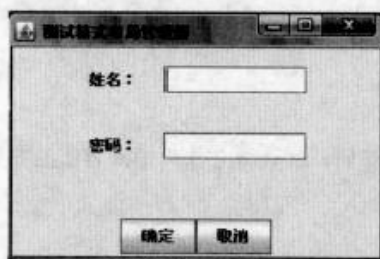


图 5.11 BoxLayout 布局管理器

其实箱式布局管理器同 GridBagLayout 布局管理器有一点相似, 并且都可以设计出很人性化的用户界面。但两者的不同点是它们将控件摆放的方式不一样, 前者为箱的形式, 后者为网格的形式。希望读者能够自行练习, 多多比较两者的不同。

5.2.7 SpringLayout

SpringLayout 布局管理器是通过定义组件的边沿距离来实现布局的。边界之间的距离是使用 Spring 对象来表示的。每一个 Spring 对象具有 4 个属性值, 包括 minimum、maximum、preferred、value, 其中 value 表示的是真实的值。

在这个布局管理器中, 涉及到如下几个常量。

- EAST: 指定组件的边界矩形的右边。
- NORTH: 指定组件的边界矩形的顶边。
- SOUTH: 指定组件的边界矩形的底边。
- WEST: 指定组件的边界矩形的左边。

在布局的时候, 经常会使用到下面的方法:

```
Void putConstraint(String e1, Component c1, int pad, String e2, Component c2);
```

上述方法用于将组件 c1 的边 e1 连接到组件 c2 的边 e2, 边与边之间的距离为 pad。下面将针对这个布局管理器设计实例, 实例代码如下所示:

```

// 这段代码主要是为读者展示如何使用 SpringLayout 布局管理器为组件进行布局
import javax.swing.*;
import java.awt.*;
// 产生不同的箱子布局管理器对象, 每个对象放置不同的控件
public class test10
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
    }
}

```



```
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jf.setVisible(true);
JPanel contentPane=new JPanel();
jf.setContentPane(contentPane);
// 创建了两个普通按钮组件、一个标签组件，将它们添加到中间容器中
JButton b1=new JButton("测试程序模块 1");
JButton b2=new JButton("测试程序模块 2");
JLabel l=new JLabel("测试程序");
contentPane.add(l);
contentPane.add(b2);
contentPane.add(b1);
// 创建一个 SpringLayout 布局管理器，并且将之作为中间容器的布局方式
SpringLayout lay=new SpringLayout();
contentPane.setLayout(lay);
// 针对每个组件设置其与边界的距离
lay.putConstraint(SpringLayout.NORTH,l, 5, SpringLayout.NORTH, contentPane);
lay.putConstraint(SpringLayout.WEST,l, 85, SpringLayout.WEST, contentPane);
lay.putConstraint(SpringLayout.EAST,l, 85, SpringLayout.EAST, contentPane);
lay.putConstraint(SpringLayout.NORTH,b1, 55, SpringLayout.NORTH, contentPane);
lay.putConstraint(SpringLayout.WEST,b1, 5, SpringLayout.WEST, contentPane);
lay.putConstraint(SpringLayout.EAST,b1, 25, SpringLayout.EAST, contentPane);
lay.putConstraint(SpringLayout.NORTH,b2, 105, SpringLayout.NORTH, contentPane);
lay.putConstraint(SpringLayout.WEST,b2, 5, SpringLayout.WEST, contentPane);
lay.putConstraint(SpringLayout.EAST,b2, 25, SpringLayout.EAST, contentPane);
}
```

上面程序代码的运行结果如图 5.12 所示。

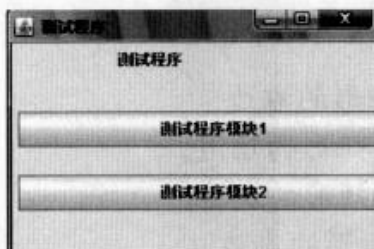


图 5.12 SpringLayout 布局管理器

这个布局管理器是通过计算组件到边的距离来给组件布局的。比较前面几个布局管理器，GridBagLayout 布局管理器和 BoxLayout 布局管理器比较人性化，而 SpringLayout 布局管理器虽然也比较人性化，但是比前面两种布局管理器要复杂的多。

5.2.8 GroupLayout

从 GroupLayout 的单词意思来看，它是以 Group（组）为单位来管理布局，也就是把多个组件（如：JLabel、JButton）按区域划分到不同的 Group（组），再根据各个 Group（组）相对于水平轴（Horizontal）和垂直轴（Vertical）的排列方式来管理。

下面将通过实例来讲述 GroupLayout 布局管理器，其代码如下所示：

```
// 这段代码主要是为读者展示如何使用 GroupLayout 布局管理器进行布局
import java.awt.Container;
import java.awt.HeadlessException;
import javax.swing.GroupLayout;
```



```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class test11 extends JFrame
{
    private static final long serialVersionUID = 1L;
    public test11() throws HeadlessException
    {
        // 创建一个中间容器, 并且创建一个 GroupLayout 布局管理器对象
        Container c = getContentPane();
        GroupLayout layout = new GroupLayout(c);
        // 创建两个普通按钮组件、文本框组件
        JButton b1 = new JButton("按钮 1");
        JButton b2 = new JButton("按钮 2");
        JTextField text = new JTextField("文本");
        // 创建一个 hsg 组, 将两个按钮一个一个的添加到组里面
        GroupLayout.SequentialGroup hsg = layout.createSequentialGroup();
        hsg.addComponent(b1);
        hsg.addComponent(b2);
        GroupLayout.ParallelGroup hpg = layout.createParallelGroup(GroupLayout.Alignment.CENTER);
        // 创建一个 hpg 组, 将文本框组件和上面的那个组添加到其中, 并且居中排列
        hpg.addComponent(text).addGroup(hsg);
        layout.setHorizontalGroup(hpg); // 沿水平线来确定 hpg 组中两个按钮组件的位置
        GroupLayout.ParallelGroup vpg = layout.createParallelGroup();
        // 创建一个 vpg 组, 按照水平线来排列两个按钮组件的位置
        vpg.addComponent(b1);
        vpg.addComponent(b2);
        GroupLayout.SequentialGroup vsg = layout.createSequentialGroup();
        // 将文本框组件和前面的容纳两个按钮组件的组同时添加到 vsg 组中
        vsg.addComponent(text).addGroup(vpg);
        // 沿垂直线来确定 vsg 中 vpg 和文本框组件的位置
        layout.setVerticalGroup(vsg);
        this.setLayout(layout);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
    }
    public static void main(String[] args)
    {
        test11 demo = new test11();
        demo.setVisible(true);
    }
}
```

上面程序代码的运行结果如图 5.13 所示。

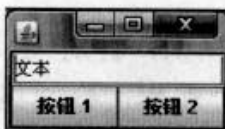


图 5.13 GroupLayout 布局管理器

上述实例先将两个按钮组件一个一个地放入到 hsg 组里, 然后将文本框组件和 hsg 放到 hpg 组中, 按照水平线对 hsg 中的两个按钮组件的位置进行排列, 最后, 再按垂直方式排列文本框组件和 hsg 组的位置。

虽然，这种布局管理器比较人性化和个性化，但是开发人员一般很少使用，因为从布局方面来说，它没有网格组布局管理器方便。

5.3 自定义布局管理器的创建

如果要创建自定义的布局管理器，就必须创建一个实现了 `LayoutManager` 接口的类，可以直接实现它或者实现它的子接口 `LayoutManager2`。每一个自定义布局管理器至少需要实现下面 5 个方法：

- `void addLayoutComponent(String, Component)`。
- `void removeLayoutComponent(Component)`。
- `Dimension preferredLayoutSize(Container)`。
- `Dimension minimumLayoutSize(Container)`。
- `void LayoutContainer(Container)`。

除了以上的 5 个方法之外，布局管理器还通常会实现至少一个公有构造函数和 `toString` 方法。如果希望支持组件布局方向、最大化、最小化、对齐方式等，还需要实现 `LayoutManager2` 接口中的 5 个方法：

- `addLayoutComponent(Component, Object)`。
- `getLayoutAlignmentX(Container)`。
- `getLayoutAlignmentY(Container)`。
- `invalidateLayout(Container)`。
- `maximumLayoutSize(Container)`。

限于篇幅的原因，在这里就不再举例说明，有兴趣的读者可以查阅相关的资料。

在实际的开发中，可以不使用布局管理器，但是这样就无法对每个控件进行定位，使得用户界面很不友好。如果有这样一种情况：一个容器所包含的组件大小不会因为容器的大小、字体、外观感觉以及语言环境的变化而变化，此时布局管理器是可以不需要的，可以使用绝对定位的方式进行定位。绝对定位适用于那种执行专门针对自身并且可能需要知道特定状态的大小、位置计算的自定义容器。但是仍旧建议读者使用布局管理器。因为使用布局管理器能够对控件进行布局，相对来说比较方便，也能够达到客户的要求。

5.4 本章小结

本章主要介绍了有关布局管理器的知识，针对 8 种不同的布局管理器给予了实例讲解，使读者能够熟悉它们的用法，也可以通过观察运行结果来比较彼此的优点和缺点，以及每个布局管理器的使用环境。读者可以将几种布局管理器结合在一起进行编程练习，以达到对多种布局管理器知识的融会贯通。

5.5 本章习题

1. 使用网格组布局管理器设计一个简单的计算器图形界面。
要求：最终的效果如图 5.14 所示。



图 5.14 使用网格组布局方式创建计算器界面

2. 使用 GridLayout 布局管理器设计一个简单的计算器图形界面。
要求：其效果如图 5.15 所示。



图 5.15 简单计算器界面

3. 设计一个学生信息窗口图形界面。
要求：其效果如图 5.16 所示。

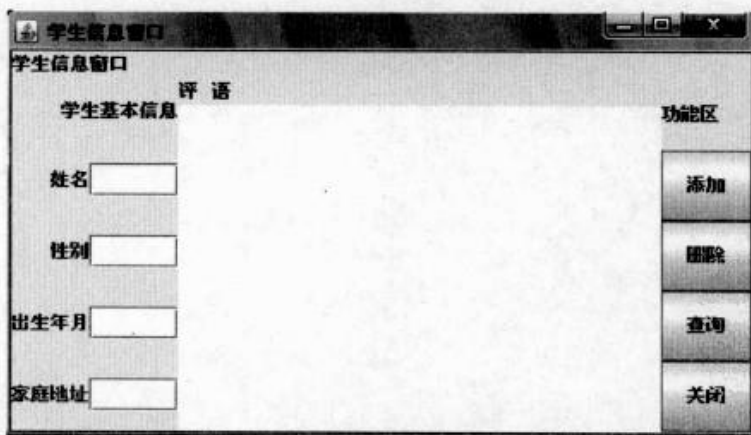


图 5.16 学生信息窗口

4. 设计一个数字翻转图形界面。
要求：
(1) 其效果如图 5.17 所示。

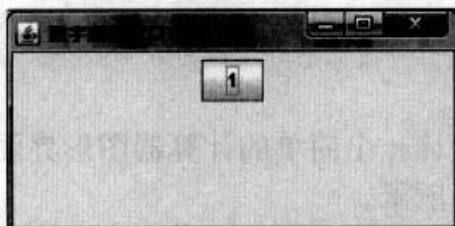



图 5.17 数字翻转 (a)

(2) 当单击  按钮时, 按钮组件上的文本变为 2, 其效果如图 5.18 所示。

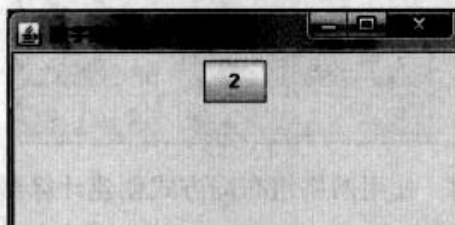


图 5.18 数字翻转 (b)



第6章 如何使用面板组件

在这里所谈论的面板，其实就是指一个容器，也就是前面所说的中间容器类，它可以将基本组件放置在其中，组成丰富多彩的用户界面，例如在前面的章节中，提到的内容面板就是一个可以放置组件的中间容器。面板里也包含了很多不同种类的面板。在实际开发中，面板的运用非常重要，同时也是不可缺少的。常用的面板有 JPanel、JScrollPane、JSplitPane、JTabbedPane、JInternalFrame、JLayeredPane 等。本章将会针对以上的各种面板进行详细分析和讲解。

通俗地说面板就是容器，准确地说就是中间容器，在这些容器内可以放置组件，例如 JScrollPane 是用于实现单个子组件的自动水平或垂直滚动的容器类。JSplitPane 用于分隔两个（只能两个）容器，两个容器图形化分隔以外观实现为基础，并且这两个 Component 可以由用户交互式调整大小。JTabbedPane 允许用户通过单击具有给定标题或图标选项卡在一组组件之间进行切换。JInternalFrame 提供很多本机窗体功能的轻量级对象，这些功能包括拖动、关闭、变成图标、调整大小、标题显示和支持菜单栏等。JLayeredPane 为 JFC、Swing 容器添加了深度，允许组件在需要时互相重叠。

面板的种类远远不止这些，以上列出的仅仅是比较常用的，也是比较典型的，将会配以程序实例进行详细讲解。

6.1 如何使用 JPanel

JPanel 是在实际项目开发中使用频率最高的面板之一。它的用处很多，有的时候会与顶层窗口相联系，有的时候还会将它放置到与顶层窗口联系的面板中。这种情况在前面的章节中也见到过，例如在前面的布局管理器中，先将一个面板放到顶层窗口中，配置好布局管理器后，再在这个布局管理器中放置几个面板。

仍然同前面章节介绍过的组件一样，如果要使用它，首先必须要通过它的构造器创建它，所以接下来将以表格的形式为读者列出 JPanel 容器类的构造器，如表 6.1 所示。

表 6.1 JPanel 容器类的构造器

JPanel 容器类的构造器	说明
JPanel()	创建一个双缓冲和流布局的新 JPanel 面板
JPanel(Boolean isDoubleBuffered)	创建具有流布局和指定缓冲策略的面板
JPanel(LayoutManager layout)	创建具有指定布局管理器的新 JPanel
JPanel(LayoutManager layout, Boolean isDoubleBuffered)	创建具有指定布局管理器和缓冲策略的面板

上表中提到的双缓冲，其意义就是通过使用双缓冲技术改进频繁变化的组件的显示效果。而所谓的流布局则是指具有 FlowLayout 特色的布局，从而可以知道默认的 JPanel 的布局管理器是 FlowLayout 布局管理器。为了能够让读者熟悉 JPanel 容器的使用，下面将通过程序实例向读者演示它的使用方法，其代码如下：

```
// 这段程序代码主要是为读者展示如何使用 JPanel 中间容器
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test1
{
    static final int WIDTH=300;
    static final int HEIGHT=150;
    public static void main(String[] args)
    {
        jf.setLayout(new BorderLayout());
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        // 创建 6 个中间容器，并且将 contentPane 放到顶层容器内
        JPanel contentPane=new JPanel();
        JPanel p1=new JPanel();
        JPanel p2=new JPanel();
        JPanel p3=new JPanel();
        JPanel p4=new JPanel();
        JPanel p5=new JPanel();
        jf.setContentPane(contentPane);
        // 创建 9 个普通按钮组件，将 p1 到 p5 个面板设置为流布局
        JButton b1=new JButton("小赵");
        JButton b2=new JButton("小李");
        JButton b3=new JButton("小王");
        JButton b4=new JButton("小孙");
        JButton b5=new JButton("小钱");
        JButton b6=new JButton("小周");
        JButton b7=new JButton("小政");
        JButton b8=new JButton("小武");
        JButton b9=new JButton("姓");
        FlowLayout fl=new FlowLayout();
        FlowLayout fl1=new FlowLayout();
        FlowLayout fl2=new FlowLayout();
        FlowLayout fl3=new FlowLayout();
        FlowLayout fl4=new FlowLayout();
        P1.setLayout(fl);
        P2.setLayout(fl1);
        P3.setLayout(fl2);
        P4.setLayout(fl3);
        P5.setLayout(fl4);
        // 将 b1、b2 加到 p1 中，将 b3、b4 加到 p2 中
        // 将 b5、b6 加到 p3 中，将 b7、b8 加到 p4 中，将 b9 加到 p5 中
        p1.add(b1);
        p1.add(b2);
        p2.add(b3);
        p2.add(b4);
```



```

p3.add(b5);
p3.add(b6);
p4.add(b7);
p4.add(b8);
p5.add(b9);
// 将 p1 到 p5 个面板按照 BorderLayout 布局方式放置到 contentPane 面板中
contentPane.add(p1,"North");
contentPane.add(p2,"South");
contentPane.add(p3,"East");
contentPane.add(p4,"West");
contentPane.add(p5,"Center");
}
}

```

上面程序代码的运行结果如图 6.1 所示。



图 6.1 JPanel 的使用

分析上面的程序，在这个程序里建立了 6 个面板容器类，分别是 contentPane、p1、p2、p3、p4、p5，其中 contentPane 是与顶层窗口关联的内容面板，其余 5 个面板容器则是与布局管理器中每个部分相关联的面板。上面的程序代码使用的是不带参数的 JPanel 容器类构造器，下面使用带参数的 JPanel 容器类构造器来重新编写上例的程序代码。其代码如下所示：

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
// 这是一个登录类。设计成一个继承容器的类
// WIDTH 是指整个顶层框架的宽度
// HEIGHT 是指整个顶层框架的长度
public class test2
{
    static final int WIDTH=300;
    static final int HEIGHT=150;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        // 创建 6 个中间容器，并且将 contentPane 放到顶层容器内
        JPanel p1=new JPanel();
        JPanel p2=new JPanel();
        JPanel p3=new JPanel();
        JPanel p4=new JPanel();
        JPanel p5=new JPanel();
        jf.setContentPane(contentPane);
        // 创建 9 个普通按钮组件，将 p1 到 p5 个面板设置为流布局
        JButton b1=new JButton("小赵");
        JButton b2=new JButton("小李");

```



```

JButton b3=new JButton("小王");
JButton b4=new JButton("小孙");
JButton b5=new JButton("小钱");
JButton b6=new JButton("小周");
JButton b7=new JButton("小政");
JButton b8=new JButton("小武");
JButton b9=new JButton("姓");
p1.setLayout(new FlowLayout());
p2.setLayout(new FlowLayout());
p3.setLayout(new FlowLayout());
p4.setLayout(new FlowLayout());
p5.setLayout(new FlowLayout());
// 将 b1、b2 加到 p1 中, 将 b3、b4 加到 p2 中
// 将 b5、b6 加到 p3 中, 将 b7、b8 加到 p4 中, 将 b9 加到 p5 中
p1.add(b1);
p1.add(b2);
p2.add(b3);
p2.add(b4);
p3.add(b5);
p3.add(b6);
p4.add(b7);
p4.add(b8);
p5.add(b9);
// 将 p1 到 p5 个面板按照 BorderLayout 布局方式放置到 contentPane 面板中
contentPane.add(p1,"North");
contentPane.add(p2,"South");
contentPane.add(p3,"East");
contentPane.add(p4,"West");
contentPane.add(p5,"Center");
}
}
```

上面的程序只是将布局管理器的设置放到构造器中, 所以其结果是一样的。希望读者能够针对剩下的两个没有讲解的构造器多多练习, 从而熟练掌握其用法。

6.2 如何使用 JScrollPane

JScrollPane 类是一个带滚动条的容器类。它可以用来显示一些文本、表格等内容。当内容超过了 JScrollPane 面板的大小时, 系统会自动添加滚动条。要想使用 JScrollPane 面板, 就必须依靠构造器, 下面将以表格的形式为读者列出 JScrollPane 面板的常用构造器, 如表 6.2 所示。

表 6.2 JScrollPane 常用构造器

JScrollPane 常用构造器	说明
public JScrollPane()	创建一个空的 JScrollPane, 需要时可将水平和垂直滚动条都显示出来
public JScrollPane(Component view)	创建一个显示指定组件内容的 JScrollPane, 只要组件的内容超过视图大小就会显示水平和垂直滚动条
public JScrollPane(Component view, int vsbPolicy,int hsbPolicy)	view 是显示在滚动窗格视口中的组件; vsbPolicy 是指定垂直滚动条策略的一个整数; hsbPolicy 是指定水平滚动条策略的一个整数, 创建一个 JScrollPane, 它将视图组件显示在一个视口中, 视图位置可使用一对滚动条控制。滚动条策略指定滚动条在何时显示

上面提到了滚动策略，所谓的滚动策略就是决定什么时候出现滚动条。构造器中的滚动策略是个整型量。下面将以表格的形式列出这些滚动策略的值以及其含义，如表 6.3 所示。

表 6.3 滚动策略变量

作用	滚动策略变量	说明
确定垂直滚动条何时显示	ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED	需要时出现垂直滚动条
	ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER	从不出现垂直滚动条
	ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS	总是出现垂直滚动条
确定水平滚动条何时显示	ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED	需要时出现水平滚动条
	ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER	从不出现水平滚动条
	ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS	总是出现水平滚动条

下面将列举一个简单的实例来说明如何使用 JScrollPane。实例代码如下所示：

```
// 这段代码主要是为读者展示如何使用 JScrollPane 组件，让文本组件具有滚动功能
import javax.swing.*;
public class test3
{
    static final int WIDTH=300;
    static final int HEIGHT=150;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JTextArea ta=new JTextArea("我们是某某软件公司的骨干开发人员，我们会竭诚为您服务!!!");
        // 创建一个文本域组件和一个滚动条面板，并且将滚动条面板添加到顶层容器内
        JScrollPane sp=new JScrollPane(ta);
        jf.setContentPane(sp);
    }
}
```

上面程序代码的运行结果如图 6.2 所示。

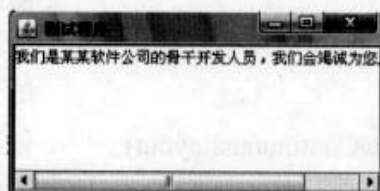


图 6.2 如何使用 JScrollPane

在以上程序中，当 JScrollPane 面板无法完全显示时才会出现滚动条。如果用户希望无论在什么时候都会显示滚动条，可将上述实例修改为如下形式：

```
// 这段程序代码主要为读者展示水平和垂直方向都会出现滚动条
import javax.swing.*;
public class test4
{
    static final int WIDTH=300;
    static final int HEIGHT=150;
    public static void main(String[] args)
```



```
{
    JFrame jf=new JFrame("测试程序");
    jf.setSize(WIDTH,HEIGHT);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setVisible(true);
    JTextArea ta=new JTextArea("我们是某某软件公司的骨干开发人员，我们会竭诚为您服务!!!");
    // 创建一个文本域组件和一个滚动条面板，并且将滚动条面板添加到顶层容器内
    JScrollPane sp=new JScrollPane(ta,ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS );
    jf.setContentPane(sp);
}
```

上面程序代码的运行结果如图 6.3 所示。

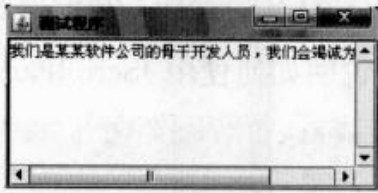


图 6.3 垂直和水平滚动条的显示

从上面两个实例可以看出，JScrollPane 面板与 JPanel 面板从原理上讲都是相同的。在实际开发工作中，程序员一般会使用 JScrollPane 面板来显示大量的文本文件，就像一个浏览器一样。

6.3 如何使用 JSplitPane

JSplitPane 面板主要用来将不同的组件分隔开来。同前面所介绍的组件一样，首先将以表格的形式给出其常用的构造器，如表 6.4 所示。

表 6.4 JSplitPane 面板常用构造器

JSplitPane 面板常用构造器	说明
JSplitPane()	创建一个默认的不带任何参数的分隔面板
JSplitPane(int newOrientation)	创建一个配置为指定方向且无连续布局的分隔面板
JSplitPane(int newOrientation,boolean newContinuousLayout)	创建一个具有指定方向和重绘方式的分隔面板
JSplitPane(int newOrientation,oolean newContinuousLayout, Component newLeftComponent,Component newRightComponent)	创建一个具有指定方向、重绘方式和指定组件的分隔面板
JSplitPane(int newOrientation,Component newLeftComponent, Component newRightComponent)	创建一个具有指定方向和不连续重绘的指定组件的面板

在上表中提到的重绘方式其实就是当分隔条改变位置时组件是否会重绘。而指定方向则是指分隔的方向是水平分隔还是垂直分隔。下面将以表格的形式列出有关 JSplitPane 类的常用方法，如表 6.5 所示。

表 6.5 JSplitPane 容器类的常用方法

JSplitPane 容器类的常用方法	说明
setOneTouchExpandable (boolean)	使该分隔面板的分隔条显示出箭头
setContinuousLayout (boolean newContinuousLayout)	设置 ContinuousLayout 属性的值, 若要使子组件连续地重新显示和布局子组件, 此值必须为 true。此属性的默认值为 false。一些外观可能不支持连续布局, 它们将忽略此属性
setOrientation(int orientation)	设置方向或者分隔窗格的方式。选项有 JSplitPane.VERTICAL_SPLIT (组件的上、下方向)、JSplitPane.HORIZONTAL_SPLIT (组件的左、右方向)
setLeftComponent(Component comp)	将组件设置到分隔条的左边 (或上面)
setRightComponent(Component comp)	将组件设置到分隔条的右边 (或下面)
setDividerSize(int newSize)	设置分隔条的大小
setDividerLocation(int location)	设置分隔条的位置
public void setOneTouchExpandable (boolean newValue)	设置 oneTouchExpandable 属性的值, 要使 JSplitPane 在分隔条上提供一个 UI 小部件来快速展开、折叠分隔条, 此属性必须为 true。此属性的默认值为 false。有些外观可能不支持一键展开, 它们将忽略此属性

下面来分析一个实例。实例程序代码如下所示:

```
// 这段程序代码主要是创建一个分隔内容面板, 将顶层容器分成两个部分
// 将两个普通按钮组件分别加到被分割的两个部分容器中
import javax.swing.*;
import java.awt.Dimension;
public class test5
{
    public static void main (String[] args)
    {
        JButton b1 = new JButton ("确定");           // 创建两个普通按钮组件
        JButton b2 = new JButton ("取消");
        JSplitPane splitPane = new JSplitPane ();     // 创建一个分隔容器类
        splitPane.setOneTouchExpandable (true);       // 让分隔线显示出箭头
        splitPane.setContinuousLayout (true);         // 当用户操作分隔线箭头时, 系统重绘图形
        splitPane.setPreferredSize (new Dimension (100,200));
        splitPane.setOrientation (JSplitPane.HORIZONTAL_SPLIT); // 设置分隔线为水平分隔线
        splitPane.setLeftComponent (b1);             // 将 b1 放到分隔线左边, 将 b2 放到分隔线右边
        splitPane.setRightComponent (b2);
        splitPane.setDividerSize (3);                // 设置分隔线大小为 3 个单位
        splitPane.setDividerLocation(50);            // 设置分隔线的位置位于中间
        JFrame frame = new JFrame ("测试窗口");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setVisible (true);
        frame.setContentPane (splitPane);
        frame.pack ();
    }
}
```

上面程序代码的运行结果如图 6.4 所示。



图 6.4 如何使用 JSplitPane 面板

以上的实例比较简单，下面给出一个稍微复杂的实例，实例代码如下所示：

```
// 这段程序代码主要是将顶层窗口分成两个部分，每一个部分添加一个带流布局的内容面板
// 再在内容面板中添加普通按钮组件
import javax.swing.*;
import java.awt.*;
public class test6
```

```
{
    public static void main (String[] args)
    {
        // 创建 4 个普通按钮组件
        JButton b1=new JButton ("确定");
        JButton b2=new JButton ("取消");
        JButton b3=new JButton("优秀");
        JButton b4=new JButton("良好");
        // 创建两个中间容器，并且设置为流布局的布局方式
        JPanel p1=new JPanel();
        JPanel p2=new JPanel();
        p1.setLayout(new FlowLayout());
        p2.setLayout(new FlowLayout());
        // 将 b1 和 b2 放到 p1 中，将 b3 和 b4 放到 p2 中
        p1.add(b1);
        p1.add(b2);
        p2.add(b3);
        p2.add(b4);
        JSplitPane splitPane = new JSplitPane ();
        splitPane.setOneTouchExpandable (true);
        splitPane.setContinuousLayout (true);
        splitPane.setPreferredSize (new Dimension (50,100));
        splitPane.setOrientation (JSplitPane.HORIZONTAL_SPLIT);
        splitPane.setLeftComponent (p1);
        splitPane.setRightComponent (p2);
        splitPane.setDividerSize (3);
        splitPane.setDividerLocation(50);
        JFrame frame = new JFrame ("测试窗口");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setVisible (true);
        frame.setContentPane (splitPane);
        frame.pack ();
    }
}
```

上面的程序代码主要是先将 JSplitPane 面板与顶层窗口进行关联，接着在面板的两边分别添加一个 JPanel 面板，再把控件分别放到这两个 JPanel 面板中去。其运行结果如图 6.5 所示。



图 6.5 JPanel 与 JSplitPane 面板的综合运用

在实际开发中，使用到 JSplitPane 面板的频率非常高，经常用来在一个窗口中显示出两种不同风格的格式。上例中是在两边分别添加按钮组件，其实还可以添加其他组件，如文本框、标签等。读者可以自行进行实验。

6.4 如何使用 JTabbedPane

JTabbedPane 面板主要用来创建选项卡容器，JTabbedPane 面板的构造器类型如表 6.6 所示。

表 6.6 JTabbedPane 面板的构造器

构造器	说明
JTabbedPane()	创建一个默认的空的选项卡对象，选项卡的布局是 JTabbedPane.Top
JtabbedPane(int tabPlacement)	创建一个指定选项卡布局的选项卡对象，其选项卡布局有 4 种：JTabbedPane.Top、JTabbedPane.Bottom、JTabbedPane.LEFT、JTabbedPane.RIGHT
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	创建一个空的 TabbedPane，使其具有指定的选项卡布局和选项卡布局策略，其布局策略有两种：JTabbedPane.WRAP_TAB_LAY 和 JTabbedPane.SCROLL_TAB_LAYOUT

创建完 JTabbedPane 面板对象后，为了能够操纵面板组件，下面将以表格的形式给出一些常用方法，如表 6.7 所示。

表 6.7 JTabbedPane 面板类的常用方法

JTabbedPane 面板类的常用方法	说明
public void setBackgroundAt(int index, Color background)	将 index 位置的背景色设置为 background，它可以为 null，在这种情况下选项卡的背景色默认为 TabbedPane 的背景色。如果在该索引位置没有选项卡，则会引发一个内部异常
public void setForegroundAt(int index, Color foreground)	将 index 位置的前景色设置为 foreground，它可以为 null，在这种情况下选项卡的前景色默认为此 TabbedPane 的前景色。如果在该索引位置没有选项卡，则会引发一个内部异常
public void setEnabledAt(int index, boolean enabled)	设置是否启用 index 位置的选项卡
public void setTitleAt(int index,String title)	将 index 位置的标题设置为 title，它可以为 null
public void setToolTipTextAt(int index, String toolTipText)	将 index 位置的工具提示文本设置为 toolTipText，它可以为 null
public void addTab(String title,Component component)	添加一个由 title 表示且没有图标 component

JTabbedPane 面板类的常用方法	说明
public void setTabPlacement(int tabPlacement)	设置此选项卡窗格的选项卡布局。可能的值为 JTabbedPane.TOP、JTabbedPane.BOTTOM、JTabbedPane.LEFT、JTabbedPane.RIGHT，如果未设置，则默认值为 SwingConstants.TOP
public void setTabLayoutPolicy(int tabLayoutPolicy)	设置在一次运行中不能放入所有的选项卡时，选项卡窗格进行布局安排的策略。可能的值为 JTabbedPane.WRAP_TAB_LAYOUT、JTabbedPane.SCROLL_TAB_LAYOUT。如果未通过 UI 进行设置，则默认值为 JTabbedPane.WRAP_TAB_LAYOUT

上表中列举出了常用的方法，接下来将根据上面所学习到的常用方法给出一个实例。实例代码如下所示：

```
// 这段程序主要是创建一个 JTabbedPane 面板，为每一个标签设置一个名称
import javax.swing.*;
import java.awt.*;
public class test7
{
    public static void main(String[] args)
    {
        try
        {
            // 显示外观风格
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e){}
        JFrame frame = new JFrame ("资金状况");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setVisible (true);           // 默认为 false
        JTabbedPane tp=new JTabbedPane();  // 创建一个选项卡容器，将之添加到顶层容器内
        frame.setContentPane(tp);
        JPanel panel1 = new JPanel ();
        JPanel panel2 = new JPanel ();
        JPanel panel3 = new JPanel ();
        JPanel panel4 = new JPanel ();
        JPanel panel5 = new JPanel ()
        // 添加选项卡容器，并且设置其中每个选项卡的标签以及其是否可启用
        tp.addTab("panel1", panel1);
        tp.setEnabledAt(0,true);
        tp.setTitleAt(0,"个人收入状况");
        tp.addTab ("panel2", panel2);
        tp.setEnabledAt (1, true);
        tp.setTitleAt (1,"工资");
        tp.addTab ("panel3", panel3);
        tp.setEnabledAt (2, true);
        tp.setTitleAt (2,"奖金");
        tp.addTab ("panel4", panel4);
        tp.setEnabledAt(0,true);
        tp.setTitleAt(3,"津贴");
        tp.addTab ("panel5", panel5);
        tp.setEnabledAt(4,true);
        tp.setTitleAt(4,"社保");
    }
}
```



```

// 设置其大小以及其选项卡的位置方向
tp.setPreferredSize (new Dimension (500,200));
tp.setTabPlacement (JTabbedPane.TOP);
// 设置选项卡在容器内的显示形式
tp.setTabLayoutPolicy (JTabbedPane.SCROLL_TAB_LAYOUT);
frame.pack();
}
}

```

上面程序代码的运行结果如图 6.6 所示。

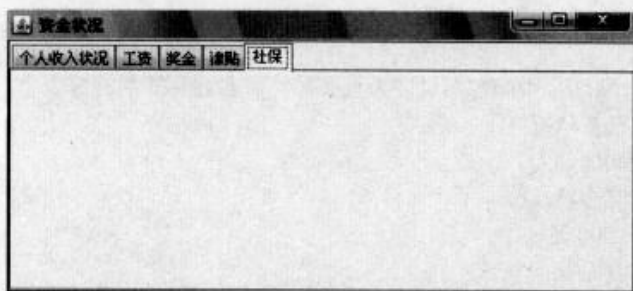


图 6.6 如何使用 JTabbedPane

上面的实例很简单，下面将在每个选项卡里添加一些组件，实例程序代码如下所示：

```

// 这段程序代码主要是在上例的基础上，在每个标签选项卡中添加基本组件
import javax.swing.*;
import java.awt.*;
public class test8
{
    public static void main(String[] args)
    {
        try
        { // 显示外观风格
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e){}
        JFrame frame = new JFrame ("资金状况");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setVisible (true); // 默认为 false
        JTabbedPane tp=new JTabbedPane(); // 创建一个选项卡容器，将之添加到顶层容器内
        frame.setContentPane(tp);
        JPanel panel1 = new JPanel ();
        JPanel panel2 = new JPanel ();
        JPanel panel3 = new JPanel ();
        JPanel panel4 = new JPanel ();
        JPanel panel5 = new JPanel ()
        // 添加选项卡容器，并且设置其中每个选项卡的标签以及其是否可启用
        tp.addTab("panel1", panel1);
        tp.setEnabledAt(0,true);
        tp.setTitleAt(0,"个人收入状况");
        tp.addTab ("panel2", panel2);
        tp.setEnabledAt (1, true);
        tp.setTitleAt (1,"工资");
        tp.addTab ("panel3", panel3);
        tp.setEnabledAt (2, true);
        tp.setTitleAt (2,"奖金");
    }
}

```



```
tp.addTab ("panel4", panel4);
tp.setEnabledAt(0,true);
tp.setTitleAt(3,"津贴");
tp.addTab ("panel5", panel5);
tp.setEnabledAt(4,true);
tp.setTitleAt(4,"社保");
// 设置其大小以及其选项卡的位置方向
tp.setPreferredSize (new Dimension (500,200));
tp.setTabPlacement (JTabbedPane.TOP);
// 设置选项卡在容器内的显示形式
tp.setTabLayoutPolicy (JTabbedPane.SCROLL_TAB_LAYOUT);
frame.pack();
// 创建 8 个标签组件, 将 5 个中间容器设置为流布局, 并且将标签组件分别放入到其中
JLabel l1=new JLabel("工资状况");
JLabel l2=new JLabel("3000 元/月");
JLabel l3=new JLabel("奖金状况");
JLabel l4=new JLabel("1500 元/月");
JLabel l5=new JLabel("津贴状况");
JLabel l6=new JLabel("500 元/月");
JLabel l7=new JLabel("社保状况");
JLabel l8=new JLabel("200 元/月");
panel2.setLayout(new FlowLayout());
panel3.setLayout(new FlowLayout());
panel4.setLayout(new FlowLayout());
panel5.setLayout(new FlowLayout());
panel2.add(l1);
panel2.add(l2);
panel3.add(l3);
panel3.add(l4);
panel4.add(l5);
panel4.add(l6);
panel5.add(l7);
panel5.add(l8);
frame.pack();
}
```

上面程序代码的运行结果如图 6.7 所示。

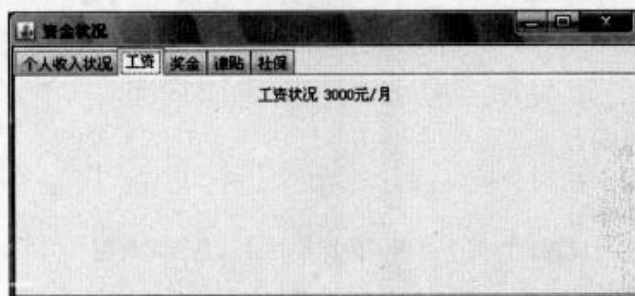


图 6.7 带布局管理器的 JTabbedPane 面板

6.5 如何使用JInternalFrame

JInternalFrame 的使用与 JFrame 几乎一样, 如最大化、最小化、关闭窗口或加入菜单等;

惟一不同的是 JInternalFrame 是中间容器类，也就是说 JInternalFrame 不能单独出现，必须依附在最上层组件中。一般会将 JInternalFrame 加入 Desktop Pane，从而方便管理，Desktop Pane 是一种特殊的 Layered Pane，用来建立虚拟桌面（Virtual Desktop），它可以显示并管理众多 Internal Frame 之间的层次关系。下面先来学习 JInternalFrame 面板的构造器，如表 6.8 所示。

表 6.8 JInternalFrame 构造器

JInternalFrame 构造器	说明
JInternalFrame()	建立一个不能更改大小、不可关闭、不可最大化且没有标题的 Internal Frame
JInternalFrame(String title)	建立一个不能更改大小、不可关闭、不可最大化但具有标题的 Internal Frame
JInternalFrame(String title,boolean resizable)	建立一个不可关闭、不可最大化但可变更大小且具有标题的 Internal Frame
JInternalFrame(String title,boolean resizable,boolean closable)	建立一个可关闭、可更改大小且具有标题，但不可最大化、最小化的 Internal Frame
JInternalFrame(String title, boolean resizable, boolean closable,boolean maximizable)	建立一个可关闭、可更改大小具有标题、可最大化，但不可最小化的 Internal Frame
JInternalFrame(String title,boolean resizable,boolean closable,boolean maximizable,boolean iconifiable)	建立一个可关闭、可更改大小、具有标题、可最大化与最小化的 Internal Frame

接下来将设计一个实例，实例代码如下：

```
// 这段程序代码主要创建两个 JInternalFrame 窗口
// 将这两个 JInternalFrame 窗口放到一个顶层容器的内容面板中
import javax.swing.*;
import java.awt.*;
public class test9
{
    static final int WIDTH=300;
    static final int HEIGHT=150;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        // 创建一个中间容器，并且将之添加到顶层容器内，将之设置为流布局
        JPanel contentPane=new JPanel();
        jf.setContentPane(contentPane);
        contentPane.setLayout(new FlowLayout());
        JDesktopPane dp=new JDesktopPane();
        // 创建一个虚拟桌面容器，将 dp 添加到以上创建的中间容器中
        dp.setLayout(new FlowLayout());
        contentPane.add(dp);
        JInternalFrame jif=new JInternalFrame("第一个窗口",true,true,true);
        // 创建两个 JInternalFrame 容器
        // 并且创建 6 个标签组件，分别将它们添加到两个 JInternalFrame 容器内
        JInternalFrame jif1=new JInternalFrame("第二个窗口",true,true,true);
        JLabel l1=new JLabel("这是我第一个窗口");
```



```
JLabel l2=new JLabel("这也是你第一个窗口");
JLabel l3=new JLabel("这同时是他第一个窗口");
JLabel l4=new JLabel("这是我第二个窗口");
JLabel l5=new JLabel("这也是你第二个窗口");
JLabel l6=new JLabel("这同时是他第二个窗口");
jif.setLayout(new FlowLayout());
jifl.setLayout(new FlowLayout());
jif.add(l1);
jif.add(l2);
jif.add(l3);
jifl.add(l4);
jifl.add(l5);
jifl.add(l6);
dp.add(jif);
dp.add(jifl);
jif.setVisible(true);
jifl.setVisible(true);
}
```

上面程序代码的运行结果如图 6.8 所示。

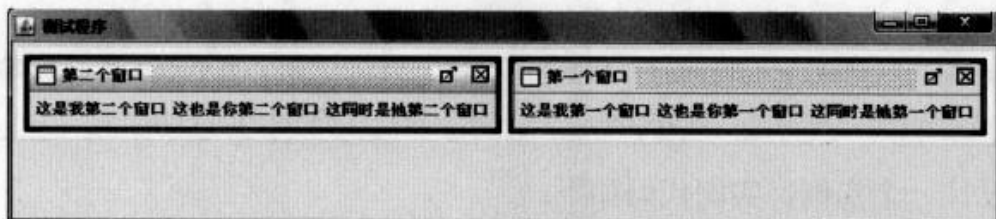


图 6.8 如何使用 JInternalFrame 面板

其实 JInternalFrame 面板的用法同 JPanel 面板几乎是相同的，惟一不同的是可以关闭、最大化、最小化面板。只要熟练掌握 JPanel 的用法，JInternalFrame 面板的用法也就不难了。

6.6 如何使用 JLayeredPane

JLayeredPane 面板主要是为 JFC、Swing 容器添加深度，它允许组件在必要的时候相互重叠。其实，JLayeredPane 面板将面板深度范围分成多个不同的层，将组件放入不同的层内，这样可以保证组件能够正确的重叠，而不必为具体的深度编号。

下面将通过实例来学习它的用法。其具体实例代码如下所示：

```
// 这段程序代码主要是将两个普通按钮组件，放置到 JLayeredPane 容器中
// 再将这两个按钮组件分成两个不同的层次，当单击下面一个层的按钮组件时，它就会被显示到上面一个层上
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test10 extends JFrame implements ActionListener
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=300;
    static final int HEIGHT=150;
    JLayeredPane lp=new JLayeredPane();
    static JButton b1=new JButton("确定");
```



```

static JButton b2=new JButton("取消");
public test10()
{
    // 设置顶层容器的标题
    super("测试窗口");
    // 将新建的 JLayeredPane 放到顶层容器内
    super.setContentPane(lp);
    b1.addActionListener(this);           // 按钮事件
    b2.addActionListener(this);
    lp.add(b1, new Integer(200));         // 将组件添加到 JLayeredPane 中, 指定所在的层
    lp.add(b2, new Integer(300));
    b1.setBounds(new Rectangle(100, 100, 100, 100)); // Button 出现位置
    b1.setVisible(true);                  // 显示
    b2.setBounds(new Rectangle(50, 50, 100, 100));
    b2.setVisible(true);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setSize(360, 260);
    this.setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
    if (e.getActionCommand().equals("确定"))
    {
        // 判断是哪个按钮的动作
        lp.setLayer(b1, 300);             // 重新设置组件层数
        lp.setLayer(b2, 200);
    }
    else if (e.getActionCommand().equals("取消"))
    {
        lp.setLayer(b1, 200);
        lp.setLayer(b2, 300);
    }
}
public static void main(String args[])
{
    new test10();
}
}

```

上面程序代码的运行结果如图 6.9 所示。

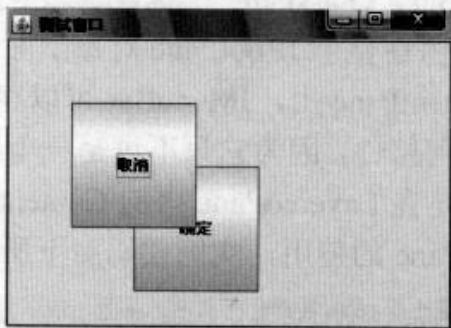


图 6.9 如何使用 JLayeredPane 面板

在上面的程序段里涉及到了一些事件侦听类的知识，这些知识将在后面的章节中进行讲解，这里不再赘述。在图 6.9 中，如果单击“确定”按钮，那么“确定”按钮会出现在最上面，如图 6.10 所示。

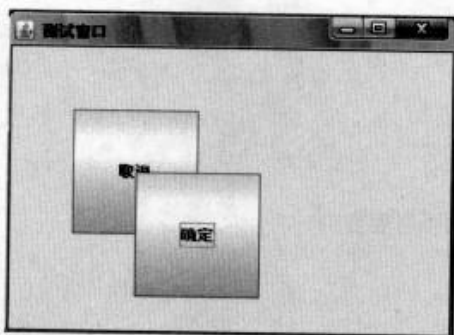


图 6.10 单击后的 JLayeredPane 面板

以上程序实例有点类似于 Windows 操作系统中的操作，几个窗口同时出现时，单击任何一个窗口都会出现在最上层。

6.7 如何使用 JRootPane

JRootPane 面板是由许多部分组成的，它可以直接从顶层容器中获得一个 JRootPane 对象来直接使用，而不需要新建一个对象。JRootPane 包括了很多部分，其层次结构如图 6.11 所示。

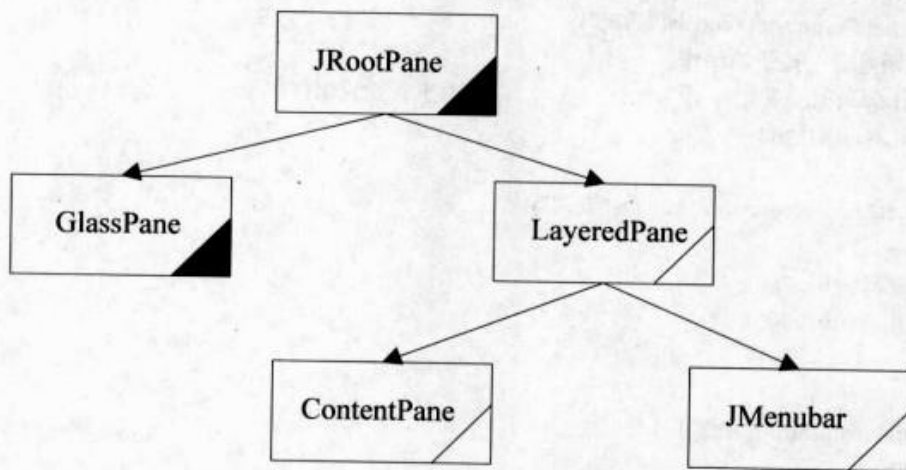


图 6.11 JRootPane 层次结构图

由上面的层次结构图可以看出，JRootPane 包括 GlassPane 和 LayeredPane，而 LayeredPane 又包括了 ContentPane 和 JMenuBar。GlassPane 在默认状态下是隐藏的，如果设置它为 true，即可见的，它就会像玻璃一样覆盖在整个 JRootPane 对象上面。而 LayeredPane 分成很多层，一般所有的组件都会添加到 ContentPane 上，JMenuBar 可以存在也可以不存在。

不能在 JRootPane 上添加任何组件，因为它只不过是一个虚拟的容器，如要在最上层添加组件，就必须在 LayeredPane 或是在 LayeredPane 内的 ContentPane 上添加。

下面将通过实例讲解 JRootPane 的使用，实例代码如下所示：

```
// 这段程序代码主要是创建一个 JRootPane 面板，然后在这个面板中添加菜单
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test11 extends JFrame
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public test11()
    {
        // ... (rest of the code)
    }
}
```



```
{
    super("测试窗口");
    // 设置顶层容器的标题, 并且将 JRootPane 设置为其依附在顶层容器上的面板
    JRootPane rp=new JRootPane();
    super.setContentPane(rp);
    JMenuBar menubar1=new JMenuBar();
    // 创建一个菜单, 并且将菜单添加到 JRootPane 中
    rp.setJMenuBar(menubar1);
    JMenu menu1=new JMenu("文件");
    JMenu menu2=new JMenu("编辑"); // 创建菜单项
    JMenu menu3=new JMenu("视图");
    JMenu menu4=new JMenu("帮助");
    menubar1.add(menu1);
    menubar1.add(menu2);
    menubar1.add(menu3);
    JMenuItem item1=new JMenuItem("打开");
    JMenuItem item2=new JMenuItem("保存");
    JMenuItem item3=new JMenuItem("打印");
    JMenuItem item4=new JMenuItem("退出");
    menu1.add(item1);
    menu1.add(item2);
    menu1.addSeparator();
    menu1.add(item3);
    menu1.addSeparator();
    menu1.add(item4);
    this.setVisible(true);
}
public static void main(String args[])
{
    new test11();
}
```

上面程序代码的运行结果如图 6.12 所示。

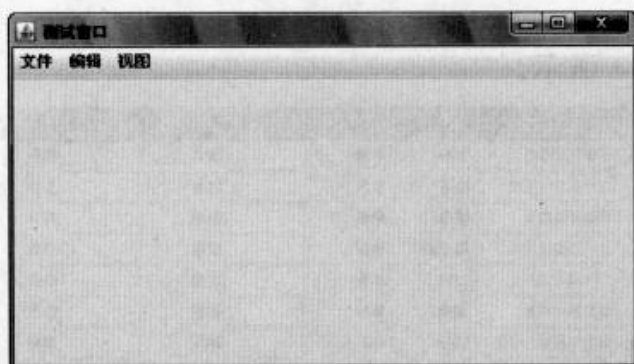


图 6.12 JRootPane 的用法

上面的程序段将菜单条添加到 JRootPane 面板中, 而在除菜单条区域外, 可以添加一个内容面板, 而后再在其中添加组件, 这样就可以实现一个完整美观的用户界面。实际上 JRootPane 只是一个容器, 是一个可以装载其他容器类的容器类, 它可以装载菜单、内容面板等。JRootPane 所使用的自定义 JLayoutManager 可确保如下内容:

- JGlassPane 填充了 JRootPane 的整个可查看区域。
- JLayeredPane 填充了 JRootPane 的整个可查看区域。

- JMenuBar 位于 LayeredPane 的上边缘。
- JContentPane 填充了整个可查看区域，除 JMenuBar（如果有）区域外。

6.8 本章小结

本章主要针对面板知识，并且结合程序实例进行详细讨论。在实际开发中，开发人员可以将本章介绍的各种面板，针对实际情况进行选择。学习本章后，读者可以结合本章的面板和上章的布局管理器开发出一些美观的用户界面。

6.9 本章习题

1. 设计一个图形界面，在这个图形界面中有一个顶层容器，在顶层容器中添加一个 JTabbedPane 面板，最后在这个面板中添加一些学生信息，例如姓名、性别、年龄、出生年月，这些信息组件按网格组布局方式布局。

要求：其效果如图 6.13 所示。

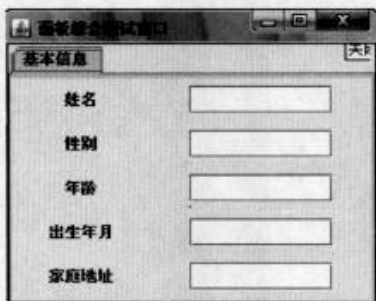


图 6.13 本章习题 1

2. 设计一个分隔面板，让面板的左边显示计算机的基本信息，面板的右边显示价格和所要订购的数量以及总价格。

要求：其效果如图 6.14 所示。



图 6.14 本章习题 2

3. 设计一个图形界面，这个图形界面的主面板是菜单和分隔面板，然后在分隔面板中，左边是一个 JTabbedPane 面板，右边是两个 JInternalFrame 面板，其中有一些与左边面板元素的相关信息。

要求：其效果如图 6.15 所示。

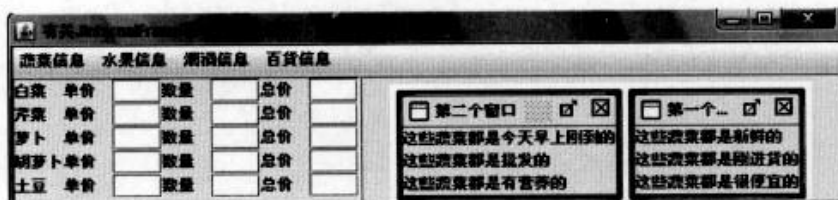


图 6.15 本章习题 3

这个题目主要考察 4 个知识点：

- 在顶层容器内添加菜单。
- 分隔面板。
- 考察 JTabbedPane，第 4 个知识点是考察 JInternalFrame 面板。

4. 将上面实例中的顶层窗口中的面板更换成 JRootPane 面板，其效果如图 6.15 所示。

第7章 Swing事件处理机制

本章主要讨论 Swing 的事件处理机制。所谓的事件处理，通俗地说就是如何处理用户软件中的事件。而所谓的事件，则是指像敲键盘、单击鼠标、双击鼠标、在文本输入区域内获取输入焦点、单击按钮控件、单击选项卡控件、单击树组件等，这些事件的发生必须要处理。当单击按钮控件后会发生什么样的事情，而这个事情就是 Swing 通过代码编写出来的。例如，在一个登录窗口，当输入密码和用户名后，单击“确定”按钮，就可以进入到内部系统了。而这个单击“确定”按钮后所发生的事情就是 Swing 要处理的。本章将会讨论如何处理以及其处理的过程。

7.1 Swing 事件处理机制概述

在学习 Swing 事件处理之前，必须先理解两个概念，分别是事件源和事件。所谓的事件就是指用户对组件发生的动作，例如，在登录到 Windows 系统时，首先要输入密码和用户名。此时，需要单击“确定”按钮才能进入，这个单击的动作就是事件。所谓的事件源就是触发动作的组件，例如，前面提到的登录窗口中的“确定”按钮就是事件源。事件处理，简单地说，就是用户要对事件源进行操作，操作了事件源就产生了事件。

明白事件和事件源的概念后，下面又要面对一个新的问题，就是用户触发事件源产生了事件，该如何处理这些事件呢？此时又要提及一个概念：事件监听者。所谓的事件监听者就是系统接收到所产生的事件，然后根据这些事件作相应的处理。有一点需要强调的是：事件源是不处理事件的，而是将事件扔给事件监听者来处理。

Swing 上的事件处理的基础其实就是 AWT 的事件处理类，AWT 组件上发生的事件对象的最终父类都是 Event Object，所以，所有的事件类都继承它。Event Object 类中有一个方法：getSource()，通过使用这个方法来获取事件源对象。

事件类的种类很多，其中包括了动作事件 (ActionEvent)、鼠标事件 (MouseEvent)、焦点事件 (FocusEvent) 等。每个事件都有不同的事件监听者，换句话说，每一种事件只能交给与之相对应的事件监听者去处理。打开 API 文档可以看到，其实每一种事件监听者都是一种接口。这就说明事件监听者中的方法都没有被实现。为什么不实现呢？事件监听者是接受事件后，根据程序员的意图去处理的。

事件类和事件监听者接口都有一定的规律。事件类都是带 Event 的，而事件监听者都是带 Listener。这样就可以很好地分辨接口的类型了。

本章将针对不同种类的事件通过实例给予详细地分析和讲解，力求让读者能够通过本章的学习对事件处理达到熟练的程度。

7.2 Swing 中的监听器

事件监听器的种类很多，正如前面所讲，每种事件对应着不同的事件监听器。Swing 所支持的事件监听器分别有：焦点监听器、键盘监听器、鼠标监听器、鼠标移动监听器、鼠标滑轮监听器、属性变化监听器等。

在实际开发中，会遇到很多事件类型，例如动作事件类、窗口事件类、文本事件类等。下面将介绍常见的事件处理方式。

7.2.1 事件处理的过程与步骤

如何针对事件进行处理呢？下面将给出事件最基本的处理过程。

- 定义实现事件监听接口类。
- 创建事件监听器。
- 向事件源注册监听器对象。

下面通过程序实例让读者熟悉一下整个事件处理的过程。这个实例主要是实现当单击按钮组件时，文本框中的文本被清空的动作事件，实例程序代码如下所示：

```
// 这段程序代码主要是创建一个文本框和一个普通按钮组件
// 当单击这个按钮组件时，会触发动作事件，清空文本框中的数据
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test1
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    static JTextField l=new JTextField(20);
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel();
        contentPane.setLayout(new BorderLayout());
        jf.setContentPane(contentPane);
        JButton b=new JButton("清空文本框中的信息");
        contentPane.add(l,"North");
        contentPane.add(b,"South");
        ActionListener ac=new ActionHandler();
        b.addActionListener(ac);
    }
}
// 定义实现事件监听类
class ActionHandler implements ActionListener
```



```
{  
    public void actionPerformed(ActionEvent e)  
    {  
        new test1().l.setText("");  
    }  
}
```

上面程序代码的运行结果如图 7.1 所示。

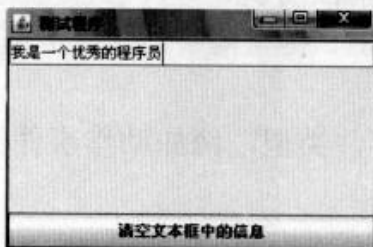


图 7.1 一个简单的事件处理程序

当单击按钮组件后，文本输入框中的信息就会被删除，其结果如图 7.2 所示。



图 7.2 事件处理后的结果

从以上的程序实例可以看出一个事件处理的过程，下面针对上面的程序进行总结。

- 定义实现事件监听接口类：class ActionHandler implements ActionListener{ }。
- 创建事件监听器：b.addActionListener(ac)。
- 向事件源注册监听器对象：ActionListener ac=new ActionHandler()。

7.2.2 匿名类方式处理事件

在实际开发中可能会出现很多个事件源，也可能出现很多个事件，如果按照上一小节的方法，就需要创建很多个事件监听器和事件监听接口类。这样会让整个程序显得非常臃肿，可读性会比较差，本节将向读者介绍一种使用匿名类的方式处理事件。

下面将通过程序实例来观察其实现的过程。这个实例是将上面的实例使用匿名类的方式进行修改，其程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何使用匿名类处理事件  
// 通过将上例进行修改，使用匿名类处理动作事件  
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
public class test2  
{  
    static final int WIDTH=300;  
    static final int HEIGHT=200;  
    static JTextField l=new JTextField(20);
```



```
public static void main(String[] args)
{
    JFrame jf=new JFrame("测试程序");
    jf.setSize(WIDTH,HEIGHT);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setVisible(true);
    JPanel contentPane=new JPanel();
    contentPane.setLayout(new BorderLayout());
    jf.setContentPane(contentPane);
    JButton b=new JButton("清空文本框中的信息");
    contentPane.add(l,"North");
    contentPane.add(b,"South");
    // 创建一个匿名类来处理按钮的动作事件
    b.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent Event)
        {
            l.setText(""); // 将文本框文本清空
        }
    });
}
```

以上程序代码的运行结果与上一小节的实例运行结果一样。这种方式主要是将注册、创建监听器、创建监听器接口类三个步骤融合在一起出现。如果整个程序出现多个事件源和事件的话，使用此方式将会大大增加程序代码的可读性。

7.2.3 适配器类

在事件监听接口中有很多方法，但是程序员并不是针对每个方法都必须去实现的，有的方法根本无须实现，但是接口有一个规定，即要实现接口，就必须实现接口中每个方法。针对这种情况，适配器类应运而生。

适配器类不需要实现一个接口中所有的方法，只须实现自己所需要的方法即可，因为适配器类会将其他不需要实现的方法自动以空实现的方式实现，例如后面学习到的窗口事件类中有 7 个方法，程序员可以任意实现其中自己所需要的方法即可，而不必将不需要实现的方法一一列出。适配器类的出现为开发人员带来了极大方便。在 Java 类库中，AWT 事件监听者接口有 7 个适配器类，如表 7.1 所示。

表 7.1 适配器类一览表

适配器类	说明
ComponentAdapter	组件适配器
ContainerAdapter	容器适配器
FocusAdapter	焦点适配器
KeyAdapter	键盘适配器
MouseAdapter	鼠标适配器
MouseMotionAdapter	鼠标移动适配器
WindowAdapter	窗口适配器

由于涉及到了具体的事件类，所以这里不再列举实例，当讨论到窗口事件时，再通过举例来证实适配器类的好处所在。

7.2.4 Swing 所支持的事件监听器

虽然 Swing 事件处理的基础是 AWT，但是并不像 AWT 那样支持所有的事件监听，例如文本编辑框中的文本监听器，在 AWT 中有，而在 Swing 中就不支持这种监听器。下面将 Swing 所支持的事件监听器以表格的方式列举出来，如表 7.2 所示。

表 7.2 Swing 所支持的事件监听器一览表

组件	动作 监听器	光标 监听器	变化 监听器	文档可撤销 编辑监听器	Item 监听器	列表选择 监听器	窗口 监听器
按钮	支持	不支持	支持	不支持	支持	不支持	不支持
复选框	支持	不支持	支持	不支持	支持	不支持	不支持
颜色选择对话框	不支持	不支持	支持	不支持	不支持	不支持	不支持
组合框	支持	不支持	不支持	不支持	支持	不支持	不支持
对话框	不支持	不支持	不支持	不支持	不支持	不支持	支持
编辑器窗格	不支持	支持	不支持	支持	不支持	不支持	不支持
文件选择对话框	支持	不支持	不支持	不支持	不支持	不支持	不支持
格式化文本框	支持	支持	不支持	支持	不支持	不支持	不支持
窗口类	不支持	不支持	不支持	不支持	不支持	不支持	支持

以上表格只列出了目前常用的组件是否支持 Swing 的事件监听器，无须死记，只须多多编写代码练习就可以熟练以上的表格。

7.2.5 窗口事件的处理

由表 7.2 可以知道窗口事件只针对在窗口对象上发生的事件，它在用户打开、关闭、最小化、最大化窗口时发生，处理窗口的事件接口是 WindowListener 接口，窗口监听器的所有方法如表 7.3 所示。

表 7.3 窗口监听器的所有方法

窗口监听器的所有方法	说明
public void windowActivated(WindowEvent e)	窗口被激活时调用的方法
Public void windowClosed(WindowEvent e)	窗口被关闭时调用的方法
Public void windowDeactivated(WindowEvent e)	窗口失去活性时调用的方法
Public void windowIconified(WindowEvent e)	窗口被最小化时调用的方法
Public void windowDeiconified(WindowEvent e)	窗口从最小化还原时调用的方法
Public void windowOpened(WindowEvent e)	窗口被打开时调用的方法

学习完窗口监听器的所有方法后，接下来将通过实例来讲解如何处理窗口事件的过程。这个实例主要用于展示当关闭一个窗口时其事件处理的过程。实例代码如下所示：

// 这段代码主要是为读者介绍如何处理窗口的事件

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
public class test3 extends JFrame
{
    public test3()
    {
        super.setTitle("测试窗口");
        // 向事件源注册监听器类
        WindowListener wh=new windowhandler();
        addWindowListener(wh);
    }
    public static void main(String[] args)
    {
        test3 me=new test3();
        me.setSize(400,300);
        me.setVisible(true);
    }
    // 创建一个窗口事件处理类
    class windowhandler implements WindowListener
    {
        public void windowActivated(WindowEvent e){} // 此方法不需要,但必须要空实现
        public void windowClosed(WindowEvent e) {} // 此方法不需要,但必须要空实现
        public void windowClosing(WindowEvent e)
        {
            JButton b1=new JButton("确定");
            JButton b2=new JButton("取消");
            JLabel l=new JLabel("你能确定关闭系统了吗? ");
            JDialog d=new JDialog((JFrame)e.getSource(),"系统出错了!",true)// 创建一个对话框
            d.setSize(200,100);
            d.setLocation(0,0);
            JPanel p=new JPanel();
            p.setLayout(new GridLayout(1,2));
            d.add(p,"South");
            d.add(l,"Center");
            p.add(b1);
            p.add(b2);
            d.setVisible(true);
            b1.setVisible(true);
            b2.setVisible(true);
            l.setVisible(true);
        }
        public void windowDeactivated(WindowEvent e){} // 此方法不需要,但必须要空实现
        public void windowIconified(WindowEvent e){} // 此方法不需要,但必须要空实现
        public void windowDeiconified(WindowEvent e){} // 此方法不需要,但必须要空实现
        public void windowOpened(WindowEvent e){} // 此方法不需要,但必须要空实现
    }
}
```

上述程序代码的运行结果如图 7.3 所示。

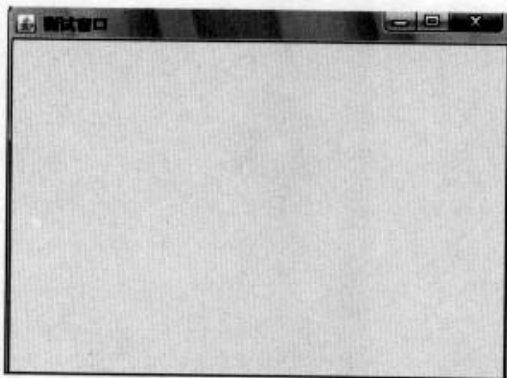



图 7.3 窗口事件类实例

当单击  按钮关闭窗口时，又会弹出一个对话框窗口，而这个对话框的弹出就是关闭事件产生的结果，如图 7.4 所示。

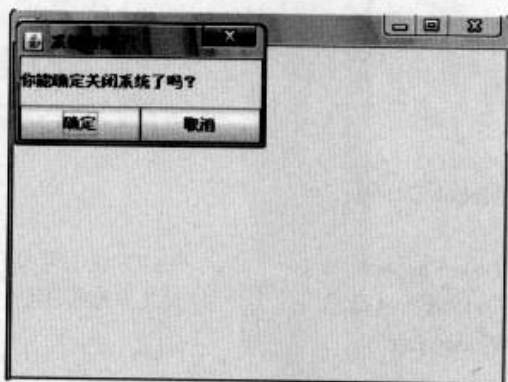


图 7.4 关闭窗口事件的结果

在适配器类一节中，谈到了窗口事件接口有 7 个没有被实现的方法，但是在上例中，只需要实现其中一个方法，而另外 6 个方法也需要空实现，这种方式在无形中给程序员带来很多不必要的麻烦。解决的方法就是可以使用适配器类。下面将通过在上例中修改观察适配器类来实现窗口事件类的过程。其程序代码如下：

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
// 创建一个窗口事件适配器类
// 在这个适配器内只需要实现关闭窗口后的事件就可以了
public class test4 extends JFrame
{
    public test4()
    {
        super.setTitle("测试窗口");
        WindowListener wh=new windowhandler();
        addWindowListener(wh);
    }
    public static void main(String[] args)
    {
        test4 me=new test4();
        me.setSize(400,300);
        me.setVisible(true);
    }
    class windowhandler extends WindowAdapter
    {
```



```

public void windowClosing(WindowEvent e)
{
    JButton b1=new JButton("确定");
    JButton b2=new JButton("取消");
    JLabel l=new JLabel("你能确定关闭系统了吗? ");
    JDialog d=new JDialog((JFrame)e.getSource(),"系统出错了!",true)    // 创建一个对话框
    d.setSize(200,100);
    d.setLocation(0,0);
    JPanel p=new JPanel();
    p.setLayout(new GridLayout(1,2));
    d.add(p,"South");
    d.add(l,"Center");
    p.add(b1);
    p.add(b2);
    d.setVisible(true);
    b1.setVisible(true);
    b2.setVisible(true);
    l.setVisible(true);
}
}

```

上面程序代码的运行结果如上例所示。在这个程序中，使用了适配器类代替了一般的窗口监听器接口类，大大减少了编写的麻烦，但是在 AWT 事件中并不是所有的事件处理监听器都有其相应的适配器类，这一点希望读者能够细细体会其原因。

7.2.6 动作事件的处理

动作事件类主要针对组件，何谓动作事件呢？例如单击按钮、选择菜单、在文本框中输入字符串并且按 Enter 键，这些都属于动作事件。动作事件监听接口是 ActionListener 接口，在这个接口中的抽象方法如下：

```
public void actionPerformed(ActionEvent e)
```

只要实现了这个方法，也就是处理了动作事件。下面给出一个实例，使读者能够熟悉动作事件处理的过程。这个实例主要是通过单击按钮组件，引起按钮的文本发生变化，实例代码如下所示：

```

// 这段程序代码主要是为读者展示如何处理动作事件，当单击按钮组件时，其上的文本会发生变化
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
// 创建一个动作事件类，在这个类中，如果单击按钮，那么按钮上的文本会发生变化
public class test5 extends Frame
{
    JButton b;
    public test5(String str)
    {
        super(str);
        b=new JButton("确认");
        add(b);
        ActionListener ac=new actionhandler();
        b.addActionListener(ac);
    }
}

```



```

    }
    public static void main(String[] args)
    {
        test5 me=new test5("动作事件测试窗口");
        me.pack();
        me.setVisible(true);
    }
    class actionhandler implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            ((JButton)e.getSource()).setLabel("取消");
        }
    }
}
    
```

上面程序代码的运行结果如图 7.5 所示。



图 7.5 动作事件处理 (a)

当单击按钮后，“确认”就会变成“取消”，如图 7.6 所示。



图 7.6 动作事件处理 (b)

动作事件的处理，在实际开发中使用的频率非常高，希望读者能从以上实例中窥视出动作事件处理的真正含义。

7.2.7 焦点事件的处理

如果在用户程序界面上有多个组件，但每次也只能操作一个组件，也就是说每次操作的焦点只能停留在一个组件上。基本上所有的组件都产生焦点事件。

焦点事件的接口中的方法主要有下面两种，如表 7.4 所示。

表 7.4 焦点事件的接口中的方法

焦点事件的接口中的方法	说明
public void focusGained(FocusEvent e)	组件获得 Focus 后被调用的方法
Public void focusLost(FocusEvent e)	组件失去 Focus 后被调用的方法

下面将给出一个实例，通过实例来讲解焦点事件的处理过程，实例程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何使用焦点事件、如何获得焦点以及如何失去焦点
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
// 此类继承了焦点监听器接口
public class test6 extends JFrame implements FocusListener
{
    List info=new List(10);
    JTextField tf=new JTextField("");
    JButton button=new JButton("确认");
    public test6(String title)
    {
        super(title);
        add(info,"North");
        add(tf,"Center");
        add(button,"South");
        f.addFocusListener(this);
    }
    public void focusGained(FocusEvent e)
    {
        if(e.isTemporary())           // 将焦点更改事件的标识为暂时性或者永久性
            info.add("暂时性获得");
        else
            info.add("长久性获得");
    }
    public void focusLost(FocusEvent e)
    {
        if(e.isTemporary())           // 将焦点更改事件的标识为暂时性或者永久性
            info.add("暂时性失去");
        else info.add("长久性失去");
    }
    public static void main(String[] args)
    {
        test6 t=new test6("测试窗口");
        t.pack();
        t.setVisible(true);
    }
}
```

上面程序代码的运行结果如图 7.7 所示。

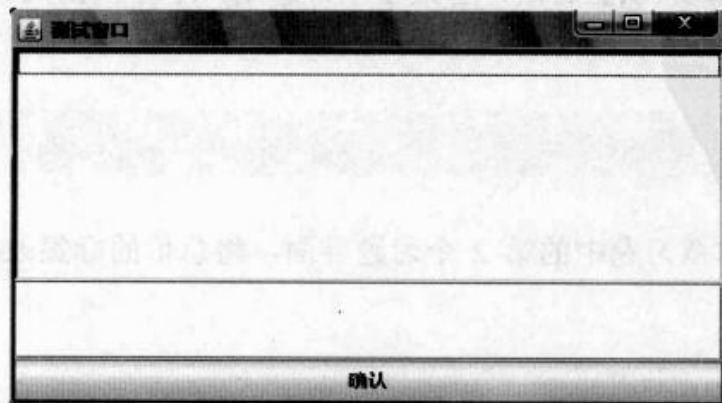


图 7.7 焦点事件的处理 (a)

当单击下面一个文本输入框时，就会弹出如图 7.8 所示界面。

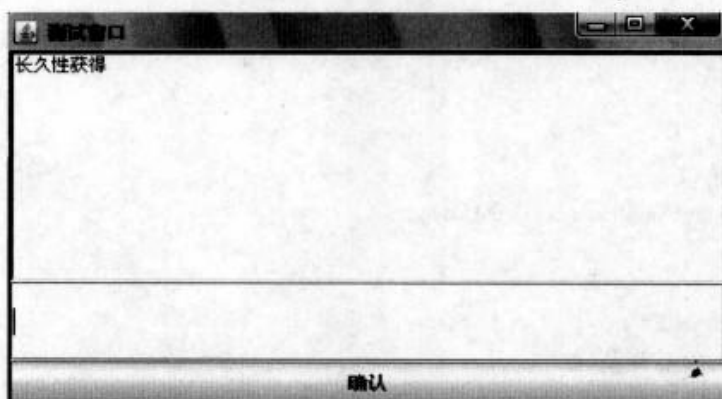


图 7.8 焦点事件的处理 (b)

当单击上面一个文本输入框时，就会弹出如图 7.9 所示界面。

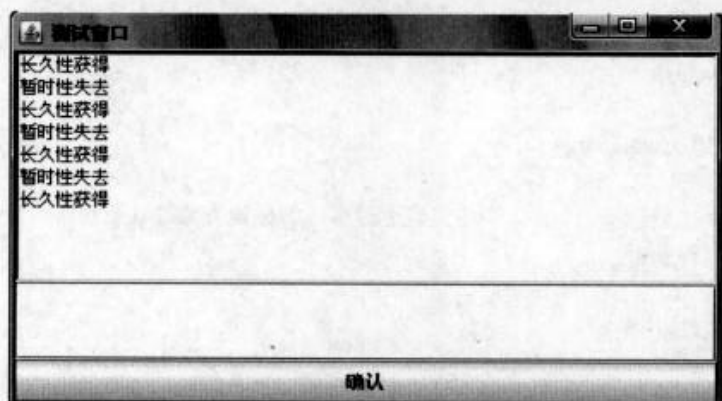


图 7.9 焦点事件的处理 (c)

上面的实例主要实现了一个焦点的获得与失去后的事件处理过程，当获得焦点时则会输出获得焦点的字样，当失去焦点时则输出失去焦点的字样。由以上实例可以看出，焦点事件处理其实很简单，实现焦点监听接口中的方法即可。

7.3 本章小结

本章主要讲述了 Swing 事件处理机制，包括事件处理的方法、适配器类、窗口事件处理、动作事件处理、焦点事件处理等。本章的重点是处理事件的方法，利用该方法大家可以进而处理其他事件，所以在学习知识的时候，必须要学好总结，学会找出同一类知识之间的共同点，找出不同知识之间的联系。

7.4 本章习题

1. 利用上一章的本章习题中的第 2 个习题界面，将总价的标签改为按钮组件，并且处理其按钮的动作事件。

要求：当在上一章第 2 个习题的界面上，将总价标签组件改为普通按钮组件后，在单价处和数量处输入数据，单击“总价”按钮组件后，会在“总价”按钮后的文本框中出现总价结果，其效果如图 7.10 所示。



图 7.10 本章习题 1

本题目主要是考察读者对于动作事件处理的掌握程度。

2. 设计一个程序，此程序中有一个菜单，单击菜单中每一个菜单项，都会产生一个动作事件，请处理这些事件。

要求：

(1) 其效果如图 7.11 所示。

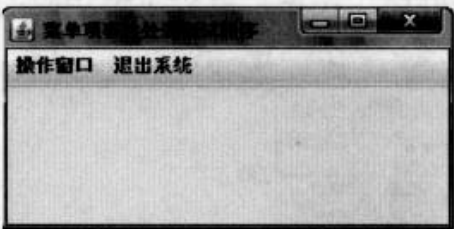


图 7.11 本章习题 2 (a)

(2) 当选择“操作窗口”|“打开窗口一”命令后，将会弹出如图 7.12 所示窗口。

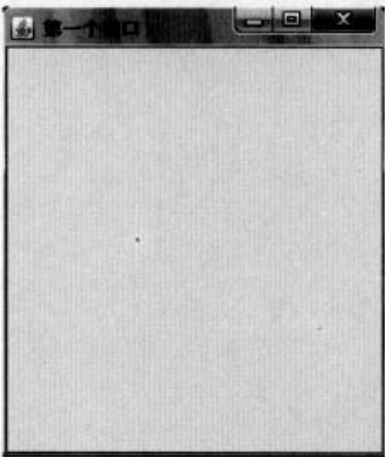


图 7.12 本章习题 2 (b)

(3) 当选择“退出系统”|“关闭窗口一”命令后，第一个窗口会自动关闭。

本题目主要是考察读者对菜单项事件处理的掌握程度。

3. 设计一个程序，此程序中有 11 个文本框，分别代表该数从 1 次方开始到 11 次方的结果，当单击“计算结果”按钮后，所有的结果都会呈现在文本框中。

要求：效果如图 7.13 所示。

原始数字	2
一次方结果	2.0
二次方结果	4.0
三次方结果	8.0
四次方结果	16.0
五次方结果	32.0
六次方结果	64.0
七次方结果	128.0
八次方结果	256.0
九次方结果	512.0
十次方结果	1024.0
计算结果	

图 7.13 本章习题 3

本题目主要是考察读者对动作事件处理的思路。

4. 设计一个程序，此程序有一个分隔面板，在面板左边有几个按钮组件，当单击每个按钮组件时，在面板的右侧会出现相应的信息。

要求：在界面中单击“我的姓名”按钮，就会出现相应的信息，其结果如图 7.14 所示。

我的姓名	王鹏
我的性别	
我的年龄	
我的工作地点	
我的家庭地址	
我的联系方式	

图 7.14 本章习题 4

第8章 如何使用列表框和下拉列表框组件

列表框和下拉列表框都是属于选择性组件，列表框是将所有选项都列在框中，以供用户选择，用户能够知道所有待选择的项。而下拉列表框则不一样，它是在用户没有操作列表框时，只显示出一个数据，当用户单击下拉箭头时，则会像卷帘门一样将数据全部展现在用户面前。两者的用途是相同的，但是两者有其使用的环境。一般来讲，当数据比较少时，可以使用列表框，但是当数据较多时，为了美观，应尽量使用下拉列表框。本章将针对这两种类似的组件结合实例给予详细讲解。

8.1 如何使用列表框 JList

列表框组件以目录的形式显示字符串，并且允许用户从中选取一项或多项，每一项被称为元素，又被称为列表项（Item）。JList 类为用户提供了可供选择的一组选项，以一系列或多列的形式显示。列表框可以有很多选项，因此它们通常被存放在同一个可以滚动的窗格内。

它有几种不同的创建方式，如数组方式、Vector 方式、ListModel 方式等。下面将为读者具体讲述每一种方式的使用。

8.1.1 使用数组方式创建列表框

使用数组的方式创建列表框，主要就是将数组中的每一项数据作为列表框中的每一项。下面将给出一个实例，从而来说明列表框 JList 的使用方法。这个实例主要是将一些数据存贮到数组中，再通过列表框的构造器——JList(String text)，来将这些数据一一列举出来，其程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何使用数组创建列表框
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class test1
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("测试窗口");
        f.setSize(400,300);
        f.setLocation(0,0);
        f.setVisible(true);
        JPanel p=new JPanel();
```



```
f.setContentPane(p);
p.setLayout(new BorderLayout());
// 将数据存储到数组 name 中
String[] name={"王鹏","王宸博","朱雪莲","王棋淋","项西云","文日珍","宋丽","田秀"};
JList l=new JList(name);
// 通过 JList(String text)构造器将数组中的数据直接列举在列表框中
JTextField tf=new JTextField();
p.add(l,"North");
p.add(tf,"South");
// 此方法的用途就是当选择列表框中的任意一项时, 都会将选择项显示在文本框中
while(true)
{
    tf.setText((String)l.getSelectedValue());
}
}
```

上面程序代码的运行结果如图 8.1 所示。

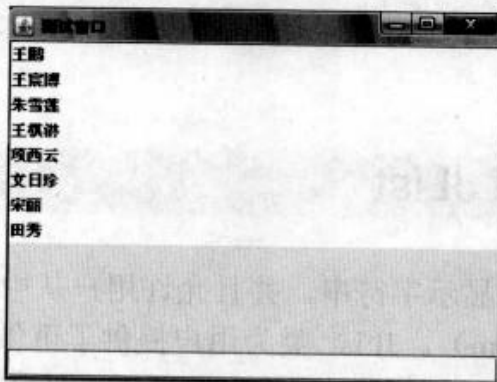


图 8.1 列表框的使用

当单击列表框中的任意一项时, 在下面的文本框中就会显示出来, 如图 8.2 所示。

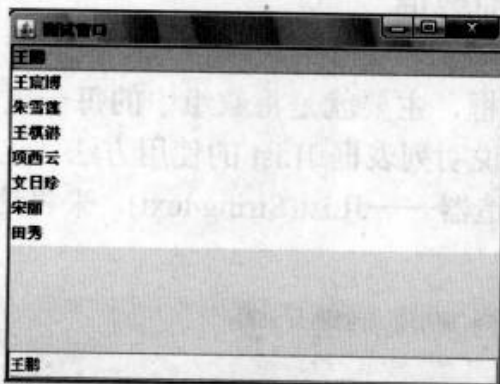


图 8.2 列表框中的选项

8.1.2 使用 Vector 方式创建列表框

除了使用数组来创建列表框外, 还可以使用 Vector 数据结构来创建列表框。使用 Vector 方式创建列表框与使用数组的方式相似, 只不过存储数据的方式不同而已。下面将通过一个实例来为读者介绍这种创建列表框的方法。

该实例主要是创建了三个列表框对象, 有两个列表框对象是通过数组的方式创建的, 而

另一个则使用 Vector 方式创建,通过此实例,读者可以比较一下两种创建列表框的方式,其程序代码如下所示:

```
// 这段程序代码主要为读者展示使用 Vector 方式和使用数组方式创建列表框的不同
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Vector;
public class test2
{
    public static void main(String args[])
    {
        JFrame f = new JFrame("JList");
        Container contentPane = f.getContentPane();
        contentPane.setLayout(new GridLayout(1,3)); // 设置面板为 GridLayout 布局方式
        // 创建两个数组, 即 s1 和 s2, 分别存储国家名和姓名
        String[] s1 = {"美国","日本","大陆","英国","法国","意大利","澳洲","韩国"};
        String[] s2 = {"范志毅","符兵","周宁","杨晨","高峰","南方","其他"};
        Vector v = new Vector(); // 创建一个 Vector 数据结构
        // 将数据存储到 Vector 数据结构中
        v.addElement("Nokia 3310");
        v.addElement("Nokia 8850");
        v.addElement("Nokia 8250");
        v.addElement("Motorola V8088");
        v.addElement("Motorola V3688x");
        v.addElement("Panasonic GD92");
        v.addElement("Panasonic GD93");
        v.addElement("NEC DB2100");
        v.addElement("Alcatel OT500");
        v.addElement("Philips Xenium 9@9");
        v.addElement("Ericsson T29sc");
        v.addElement("其他");
        JList list1 = new JList(s1); // 创建一个列表框
        // 设置主题
        list1.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
        JList list2 = new JList(s2); // 创建一个列表框
        // 一次只能选择一个列表索引
        list2.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        list2.setBorder(BorderFactory.createTitledBorder("您最喜欢哪个运动员呢?")); // 设置主题
        JList list3 = new JList(v); // 创建一个列表框
        // 一次只能选择一个列表索引
        list3.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
        list3.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一种手机?"));
        contentPane.add(new JScrollPane(list1));
        contentPane.add(new JScrollPane(list2));
        contentPane.add(new JScrollPane(list3));
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
}
```




以上程序段的运行结果如图 8.3 所示。



图 8.3 使用 Vector 创建列表框

上面的程序首先建立一个 Vector 数据结构对象，将数据存储在对象中，然后将这个对象显示到列表框中。

8.1.3 使用 ListModel 方式创建列表框

接下来将使用 ListModel 方法创建列表框。其实，ListModel 是一个接口，主要的功能是定义一些方法，让 JList 组件取得每个项目的值，并可限定项目的显示时间与方式，下面将以表格的形式列举出 ListModel 接口所定义的方法，如表 8.1 所示。

表 8.1 ListModel 接口定义的方法

定义的方法	方法的含义
void addListDataListener(ListDataListener l)	当 data model 的长度或内容值有任何改变时，利用此方法就可以处理 ListDataListener 的事件。data model 是 Vector 或 Array 的数据类型，里面存放 List 中的值
Object getElementAt(int index)	返回在 index 位置的 Item 值
int getSize()	返回 List 的长度
void removeListDataListener(ListDataListener l)	删除 ListDataListener

必须实现 ListModel 接口中所有的方法，才能利用构造器建立 JList。然而，如果要想实现 ListModel 所有的方法有些麻烦，因为一般不会用 addListDataListener()和 removeListDataListener()这两个方法。因此，Java 提供了 AbstractListModel 抽象类，此抽象类已经实现了 addListDataListener()与 removeListDataListener()两个方法。若要继承 AbstractListModel 类，就无须实现这两个方法，只须实现 getElementAt()与 getSize()方法即可，下面将列举一个实例，这个实例将通过创建一个实现了抽象类 AbstractListModel 的 DataModel 类来创建一个列表框，其代码如下所示：

```
// 这段程序代码主要是为读者展示如何使用 AbstractListModel 抽象类创建列表框
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class test3
{
    public test3()
    {
        JFrame f=new JFrame("JList");
        Container contentPane=f.getContentPane();
```



```

ListModel mode=new DataModel();
JList list=new JList(mode);           // 利用 ListModel 建立一个 JList
list.setVisibleRowCount(5);          // 设置程序一打开时所能看到的数据项个数
list.setBorder(BorderFactory.createTitledBorder("配置一台电脑需要的组件"));
contentPane.add(new JScrollPane(list));
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    new test3();
}
}

class DataModel extends AbstractListModel
// 创建一个类，实现抽象类 AbstractListModel，用来创建一个列表框
{
    String[] s={"主板","显示器","内存","CPU","硬盘","电源","键盘","鼠标"};
    public Object getElementAt(int index)
    {
        return (index+1)+ "." +s[index++];
    }
    public int getSize()
    {
        return s.length;
    }
}

```

上面程序段的运行结果如图 8.4 所示。

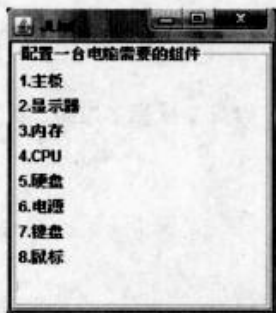


图 8.4 使用 ListModel 方法创建 JList

实际上，Java 本身还提供了另一个类，即 DefaultListModel 实体类。此类继承了 AbstractListModel 抽象类，并实现里面所有的抽象方法，因此，不需要再让程序员自己实现任何方法，可以说是相当方便。DefaultListModel 类提供不少好用的方法，例如可以随意增加一个项目（addElement()）或是删除一个项目（removeElement），甚至可以很方便地做到查询（getElementAt()）与汇出（copyInto()）项目的操作，另外利用 DefaultListModel 可以直接动态

地更改 JList 的项目值,而不需要自行产生一个 Vector 对象,相对于 JList(Vector v)构造函数而言,更方便且实用。下面将给出一个实例,通过创建一个继承 DefaultListModel 的类来创建一个列表框。其具体的程序代码如下所示:

```
// 这段程序主要是使用继承 DefaultListModel 的类来创建一个列表框
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Vector;
public class test4
{
    public test4()
    {
        JFrame f = new JFrame("JList");
        Container contentPane = f.getContentPane();
        ListModel mode = new DataModel();
        JList list = new JList(mode);
        // 通过创建一个 DefaultListModel 类的继承类来创建一个列表框对象
        list.setVisibleRowCount(5);
        list.setBorder(BorderFactory.createTitledBorder("电脑配件"));
        contentPane.add(new JScrollPane(list));
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String args[])
    {
        new test4();
    }
}

class DataModel extends DefaultListModel // 创建一个继承了 DefaultListModel 类的类
{
    String[] s = {"显示器","主板","硬盘","内存","键盘","鼠标","电源","光驱"};

    DataModel()
    {
        for(int i=0; i < s.length; i++)
            addElement((i+1)+". "+s[i]);
    }
}
```

上面程序代码的运行结果如图 8.5 所示。

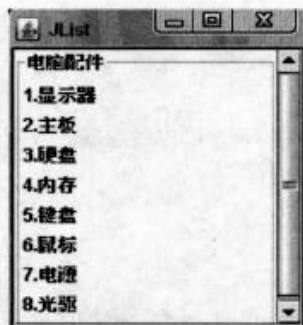


图 8.5 使用 DefaultListModel 类的继承类创建列表框

8.1.4 列表框选取事件的处理

在 JList 类中有 addListSelectionListener() 方法，可以检测用户是否对 JList 的选取有任何的改变。在 ListSelectionListener 接口中只定义一个方法，那就是 valueChanged(ListSelectionEvent)，所以必须实现这个方法，才能在用户改变选取值时取得用户最后的选取状态。下面将给出一个实例，让读者能够学习到如何在列表框中处理事件。

这个程序首先使用数组创建了一个列表框对象，然后再通过实现列表框的事件处理方法，让列表框中选项被选中时在标签组件中显现出来，其代码程序如下所示：

```
// 这段程序代码是创建一个列表框，单击列表框中某个选项后，标签组件会显示所选选项
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
// 由于 ListSelectionEvent 是 Swing 的事件，不是 AWT 的事件，因此必须引入 javax.swing.event.*
public class test5 implements ListSelectionListener
{
    JList list = null;
    JLabel label = null;
    String[] s = {"美国", "日本", "大陆", "英国", "法国", "意大利", "澳洲", "韩国"};
    public test5()
    {
        JFrame f = new JFrame("JList");
        Container contentPane = f.getContentPane();
        contentPane.setLayout(new BorderLayout());
        label = new JLabel();
        list = new JList(s); // 利用数据创建列表框
        list.setVisibleRowCount(5);
        list.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢? "));
        list.addListSelectionListener(this);
        contentPane.add(label, BorderLayout.NORTH);
        contentPane.add(new JScrollPane(list), BorderLayout.CENTER);
        f.pack();
        f.show();
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String args[])
    {
        test5 t = new test5();
        t.test5();
    }
}
```



```

{
    new test5();
}
// 实现 valueChanged()方法, 通过这个方法将列表框中所选取的数据显示在标签中
public void valueChanged(ListSelectionEvent e)
{
    int tmp = 0;
    String stmp = "您目前选取: ";
    int[] index = list.getSelectedIndices();    // 所选数据的序列号
    for(int i=0; i < index.length; i++)
    {
        tmp = index[i];
        stmp = stmp+s[tmp]+" ";
    }
    label.setText(stmp);                      // 为标签赋值
}
}

```

上面程序段的运行结果如图 8.6 所示。



图 8.6 JList 的选取事件处理

当任意选择一项时, 就会在 Label 上显示出来, 这样就完成了选择事件的处理过程, 如单击“美国”选项时效果如图 8.7 所示。

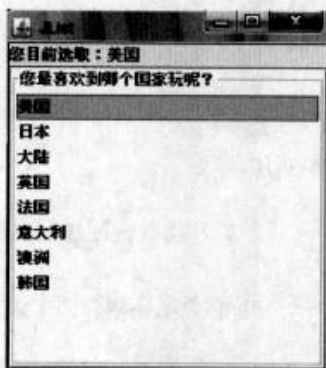


图 8.7 选取事件的结果

8.1.5 列表框双击事件的处理

列表框还有一个常用的事件, 就是如何处理在 JList 上双击鼠标的操作。其实, JList 本身没有提供 EventListener 监听器, 为了能处理双击操作, 必须利用鼠标监听器来捕获双击鼠标。但是如何知道在哪个 Item 上双击鼠标呢? 可以使用 JList 类提供的 LocatToindex()方法获知。

下面将给出实例来讲解如何处理列表框的双击事件。这个程序首先创建一个 DefaultListModel 类的继承类，然后将其实例化成两个不同的列表框，当双击左边的列表框中的数据时，就会在右边添加相应的数据，反之亦然。其具体的程序代码如下所示：

```
// 这段程序代码是创建两个列表框，然后当双击左边的列表框的某项时
// 此项将会从左边列表框移动到右边列表框
// 如果双击右边列表框中的某项，此项将会从右边移动到左边列表框
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class test6 extends MouseAdapter
{
    JList list1=null;
    JList list2=null;
    DefaultListModel model1=null;
    DefaultListModel model2=null;
    String[] s = {"Java 入门书","C 语言书","Hibernate 书","Visual Basic 书","ASP 书","Dephi 书","电脑组装书","Photoshop 书"};
    public test6()
    {
        JFrame f=new JFrame("JList");
        Container contentPane=f.getContentPane();
        contentPane.setLayout(new GridLayout(1,2));
        model1=new DataModel(1);
        list1=new JList(model1);
        list1.setBorder(BorderFactory.createTitledBorder("图书种类!"));
        list1.addMouseListener(this);// 一遇到鼠标事件立即执行.
        model2=new DataModel(2);
        list2=new JList(model2);
        list2.setBorder(BorderFactory.createTitledBorder("你所需要选择的书: "));
        list2.addMouseListener(this);// 一遇到鼠标事件立即执行.
        contentPane.add(new JScrollPane(list1));
        contentPane.add(new JScrollPane(list2));
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        new test6();
    }
    // 处理鼠标双击事件
    public void mouseClicked(MouseEvent e)
    {
        int index;
        if (e.getSource()==list1)
        {
            if (e.getClickCount()==2)
```



```

    {
        // 当双击左边列表框中选项时, 会在左边将此项去掉, 在右边列表框中将此项添加
        index=list1.locationToIndex(e.getPoint());
        String tmp=(String)model1.getElementAt(index);
        mode2.addElement(tmp);
        list2.setModel(mode2);
        model1.removeElementAt(index);
        list1.setModel(mode1);
    }
}
if (e.getSource()==list2)
{
    if (e.getClickCount()==2)
    {
        // 当双击右边列表框中选项时, 会在右边将此项去掉, 在左边列表框中将此项添加
        index=list2.locationToIndex(e.getPoint());
        String tmp=(String)mode2.getElementAt(index);
        mode1.addElement(tmp);
        list1.setModel(mode1);
        mode2.removeElementAt(index);
        list2.setModel(mode2);
    }
}
}
// 让 DataModel 继承 DefaultListModel 类, 创建一个列表框类
class DataModel extends DefaultListModel
{
    DataModel(int flag)
    {
        if (flag==1)
        {
            for (int i=0;i<s.length;i++)
                addElement(s[i]);
        }
    }
}
}

```

上面程序段的运行结果如图 8.8 所示。

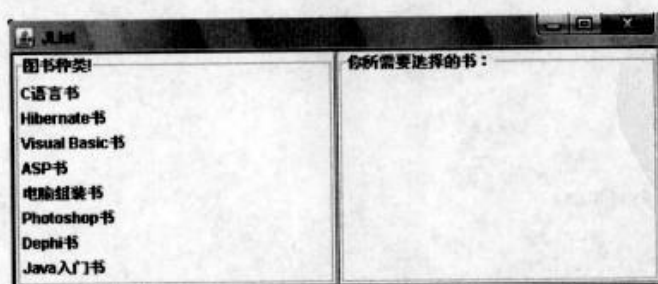


图 8.8 列表框的双击事件

双击图中左边列表框的任意项后, 此项将左边列表框移动到右边列表框中, 其结果如图 8.9 所示。

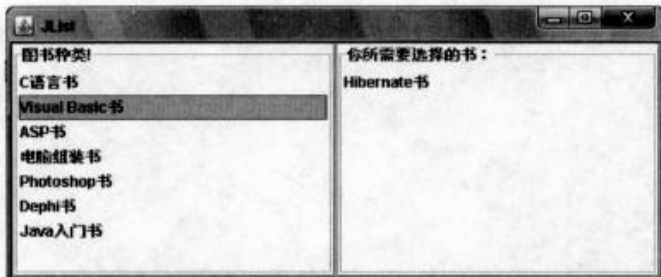


图 8.9 列表框双击后的结果

以上所讨论的就是在开发中列表框的常用知识。其实还有很多知识在这里没有讨论，希望有兴趣的读者可以参考相关的专业资料。

8.2 如何使用下拉列表框 JComboBox

下拉列表框与列表框的使用方式一样，只不过下拉列表框是将列表中所有的选项隐藏在下拉框中，显示出来的是用户需要的数据。下面将通过表格的形式列出下拉列表框的构造器，如表 8.2 所示。

表 8.2 下拉列表框的构造器

下拉列表框构造器	说明
JComboBox()	建立一个新的 JComboBox 组件
JcomboBox(ComboBoxModel dataModel)	利用 ComboBoxModel 建立一个新的 JComboBox 组件
JComboBox (Object[]ComboBoxData)	利用 Array 对象建立一个新的 JComboBox 组件
JComboBox (Vector listData)	利用 Vector 对象建立一个新的 JComboBox 组件

同样，它也与 JList 一样，可以通过数组和 Vector 来创建下拉列表框对象，下面将通过实例来详细讲述创建方法。

8.2.1 使用数组和 Vector 创建下拉列表框

使用数组和 Vector 方式创建下拉列表框，下面将通过实例来分析两种列表框的创建方式。

以下程序创建了两个下拉列表框，其中一个是使用数组创建的，另一个是使用 Vector 数据结构来创建的。其实例程序代码如下所示：

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Vector;

public class test7
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("JComboBox1");
        Container contentPane=f.getContentPane();
        contentPane.setLayout(new GridLayout(1,2));
        String[] s = {"桃花","梅花","玫瑰","牡丹","月季","茉莉","菊花","樱花"};
        // 创建一个数组，用来构造下拉列表框
        Vector v=new Vector();
        // 创建一个 Vector，用来构造下拉列表框
```



```
// 往 Vector 中添加元素
v.addElement("王鹏");
v.addElement("谭妮");
v.addElement("朱敏");
v.addElement("宋兵");
v.addElement("李丽");
v.addElement("章儒");
JComboBox combo1=new JComboBox(s);           // 利用数组创建下拉列表框
combo1.addItem("映山红");
// 利用 JComboBox 类所提供的 addItem()方法, 加入一个项目到此 JComboBox 中
combo1.setBorder(BorderFactory.createTitledBorder("你最喜欢哪种花?"));
JComboBox combo2=new JComboBox(v);           // 利用 Vector 创建下拉列表框
combo2.setBorder(BorderFactory.createTitledBorder("你最好的朋友呢? "));
contentPane.add(combo1);
contentPane.add(combo2);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
```

上面程序的运行结果如图 8.10 所示。

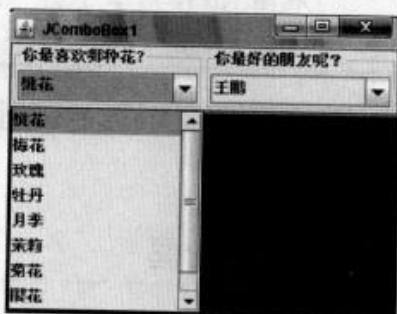


图 8.10 使用数组和 Vector 创建下拉列表框

由以上程序代码可以看出, 下拉列表框在使用 Vector 和数组创建时几乎差不多。另外, JComboBox 控件有 addItem 方法, 而这个方法在 JList 中是不具备的。

8.2.2 使用 ComboBoxModel 创建下拉列表框

同 ListModel 一样, ComboBoxModel 是一个接口, 里面定义了两个方法, 即 setSelectedItem() 与 getSelectedItem()。这两个方法的目的是让用户选取某个项目后, 可正确地显示出用户所选取的项目。JComboBox 是利用 ComboBoxModel 实现的, 有点类似于 ListModel。不过 ComboBoxModel 接口是继承自 ListModel 接口, 如果要利用 ComboBoxModel 来构造 JComboBox, 除了要实现 ComboBoxModel 的两个方法外, 还必须实现 ListModel 定义的 4 个方法, 这种做法可以说是相当麻烦。

在介绍 JList 时曾经提到 AbstractListModel 抽象类。这个抽象类实现了 ListModel 接口中

的 `addListDataListener()` 和 `removeListDataListener()` 两个方法。因此，如果继承 `AbstractListModel`，则不用实现这两个方法，只须实现 `getElementAt()`、`getSize()`、`setSelectedItem()` 与 `getSelectedItem()` 共 4 个方法。这种方法比较简单。

下面将给出一个实例，其代码如下所示：

```
// 这段代码是利用 ComboBoxModel 来创建下拉列表框
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class test8
{
    String[] s= {"王鹏","朱雪莲","王宸博","朱广兴","朱广莲","马力","欧海","黎明"};
    public test8()
    {
        JFrame f=new JFrame("JComboBox");
        Container contentPane=f.getContentPane();
        ComboBoxModel mode=new UserDefineComboBoxModel();
        // 创建一个 UserDefineComboBoxModel 对象
        JComboBox combo=new JComboBox(mode);
        // 通过 UserDefineComboBoxModel 对象来创建一个下拉列表框
        combo.setBorder(BorderFactory.createTitledBorder("你的好朋友是谁?"));
        contentPane.add(combo);
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        new test8();
    }
    // 创建一个继承 AbstractListModel 的同时
    // 实现 ComboBoxModel 这个接口的类 UserDefineComboBoxModel
    class UserDefineComboBoxModel extends AbstractListModel implements ComboBoxModel
    {
        String item=null;
        public Object getElementAt(int index)
        // 由于继承 AbstractListModel 抽象类
        // 因此分别在程序中实现了 getElementAt()与 getSize()方法
        {
            return s[index++];
        }
        public int getSize()
        {
            return s.length;
        }
        public void setSelectedItem(Object anItem)
        // 由于实现了 ComboBoxModel interface
        // 因此必须在程序中实现 setSelectedItem()与 getSelectedItem()方法
        {

```



```

        item=(String)anItem;
    }
    public Object getSelectedItem()
    {
        return item;
    }
}

```

上面程序代码的运行结果如图 8.11 所示。

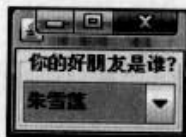


图 8.11 使用 JComboBox 创建下拉列表框

同 JList 一样, Java 对于 JComboBox 也提供了另一个类, 即 DefaultComboBoxModel 实体类。此类继承了 AbstractListModel 抽象类, 也实现了 ComboBoxModel 接口, 因此不需要再实现 getSize()、getElementAt()、setSelectedItem() 与 getSelectedItem() 方法。利用 DefaultComboBoxModel 类可以很方便地做到动态更改 JComboBox 的项目值。当没有必要自己定义特殊的 ComboBoxModel 时, 使用 DefaultComboBoxModel 就显得非常方便。下面将通过一个实例来讲解这种创建方法, 其程序代码如下所示:

```

// 这段程序代码是利用 DefaultComboBoxModel 模型创建一个下拉列表框
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class test9
{
    String[] s = {"主板", "硬盘", "内存", "电源", "键盘", "鼠标", "显示器", "光驱"};
    public test9()
    {
        JFrame f=new JFrame("JComboBox3");
        Container contentPane=f.getContentPane();
        ComboBoxModel mode=new AModel();           // 创建一个 ComboBoxModel 对象
        JComboBox combo=new JComboBox(mode);        // 使用上面的对象来创建一个下拉列表框
        combo.setBorder(BorderFactory.createTitledBorder("电脑配件"));
        contentPane.add(combo);
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        new test9();
    }
    // 利用继承 DefaultComboBoxModel 类创建一个用来建立下拉列表框对象的类
    class AModel extends DefaultComboBoxModel

```



```

{
    AModel(String[] s)
    {
        for (int i=0;i<s.length;i++)
            addElement(s[i]);
    }
}

```

上面程序的运行结果如图 8.12 所示。

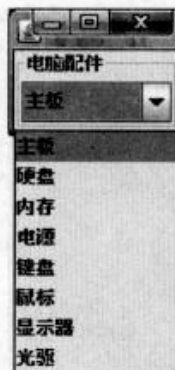


图 8.12 使用 DefaultComboBoxModel 创建下拉列表框

通过上述所有方法的比较可知，DefaultComboBoxModel 是最简单、最方便的创建下拉列表框的方法，希望读者能够多多练习比较其中的奥妙。

8.2.3 下拉列表框的事件处理

JComboBox 的事件处理也可分为两种，一种是取得用户选取的项目；另一种是用户在 JComboBox 上自行输入完毕后按下回车键，运行相对应的工作。对于第一种事件的处理，可使用 ItemListener 实现。对于第二种事件的处理，可使用 ActionListener 实现。

下面将给出一个实例，该实例主要通过选择下拉列表框中的数字来改变字体的大小，其具体的程序代码如下所示：

```

// 这段程序代码是创建一个下拉列表框，然后通过选择下拉列表框中的数字来改变字体的大小
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class test10 implements ItemListener,ActionListener
{
    String[] fontsize={"12","14","16","18","20","22","24","26","28"};
    String defaultMessage="请选择或直接输入文字大小!";
    Font font=null;
    JComboBox combo=null;
    JLabel label=null;
    public test10()
    {
        JFrame f=new JFrame("JComboBox");
        Container contentPane=f.getContentPane();
        contentPane.setLayout(new GridLayout(2,1));
        label=new JLabel("Swing",JLabel.CENTER);
        font=new Font("SansSerif",Font.PLAIN,12);
        label.setFont(font);
        combo=new JComboBox(fontsize);
    }
}

```



```

combo.setBorder(BorderFactory.createTitledBorder("请选择你要的文字大小:"));
combo.setEditable(true);
comboBoxEditor editor=combo.getEditor();
combo.configureEditor(editor,defaultMessage);
// 返回用于绘制和编辑 JComboBox 字段中所选项的编辑器
combo.addItemListener(this);
combo.addActionListener(this);
contentPane.add(label);
contentPane.add(combo);
f.pack();
f.show();
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    new test10();
}
public void actionPerformed(ActionEvent e)
{
    boolean isaddItem=true;
    int fontsize=0;
    String tmp=(String)combo.getSelectedItem();
    try
    { // 判断用户所输入的项目是否有重复, 若有重复则不增加到 JComboBox 中
        fontsize=Integer.parseInt(tmp);
        for(int i=0;i<combo.getItemCount();i++)
        {
            if (combo.getItemAt(i).equals(tmp))
            {
                isaddItem=false;
                break;
            }
        }
        if (isaddItem)
        {
            combo.insertItemAt(tmp,0);
            // 插入项目 tmp 到 0 索引位置 (第一列中)
        }
        font=new Font("SansSerif",Font.PLAIN,fontsize);
        label.setFont(font);
    }
    catch(NumberFormatException ne)
    {
        combo.getEditor().setItem("你输入的值不是整数, 请重新输入!");
    }
}
public void itemStateChanged(ItemEvent e)
// ItemListener 界面只有 itemStateChanged()一个方法, 在此实现它
{
    if (e.getStateChange()==ItemEvent.SELECTED)
        // 当用户的选择改变时, 则在 JLabel 上会显示出 Swing 目前字形大小信息
    {
        int fontsize=0;
    }
}

```



```
try
{
    fontsize=Integer.parseInt((String)e.getItem());
    label.setText("Swing 目前字形大小:"+fontsize);
}
catch(NumberFormatException ne)
{
    // 若所输入的值不是整数, 则不作任何操作
}
}
```

上面程序代码的运行结果如图 8.13 所示。

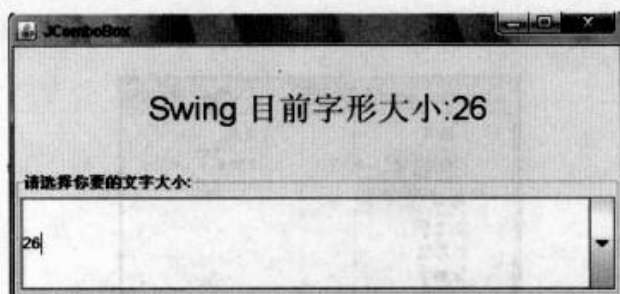


图 8.13 下拉列表框的事件处理 (a)

当选择不同的文字大小时, 那么“Swing 目前字形大小:”文本框将会根据下面文字大小而改变, 如图 8.14 所示。

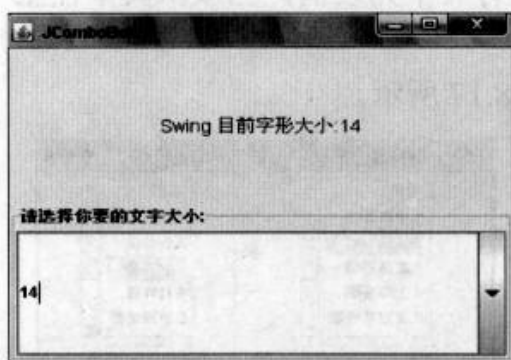


图 8.14 下拉列表框的处理 (b)

在此下拉列表框中, 还可以自行输入信息来改变文字的大小。

8.3 本章小结

本章主要是针对列表框和下拉列表框的常用构造器、事件处理方法等, 通过大量的实例让读者有了一个比较清晰的概念。在每一节中只是讲述了目前比较常用的一些方法, 当然, 还有很多其他的方法都没有讲解, 希望读者能够根据本书介绍的学习方法继续学习其他知识。

8.4 本章习题

1. 利用 DefaultListModel 方式设计两个列表框, 列表框中显示各种各样的香烟和名酒。要求: 运行后的最终效果如图 8.15 所示。

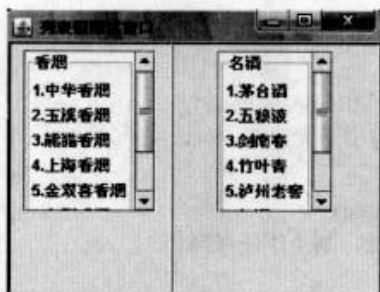


图 8.15 本章习题 1

2. 利用 DefaultComboBoxModel 方式设计两个下拉列表框, 下拉列表框中有各种各样的蔬菜和食品。

要求: 运行的最终效果如图 8.16 所示。

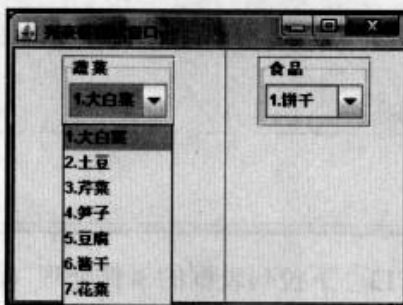


图 8.16 本章习题 2

3. 设计一个程序, 程序中有一个列表框, 当双击其中任意一项时, 在文本框中都会显示相应的数据。

要求: 运行后的效果如图 8.17 所示。

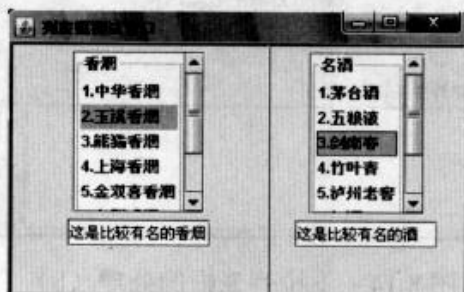


图 8.17 本章习题 3

4. 设计一个程序, 程序中有一个下拉列表框, 当选中其中任意一项时, 在文本框中都会显示相应的数据。

要求: 其效果如图 8.18 所示。

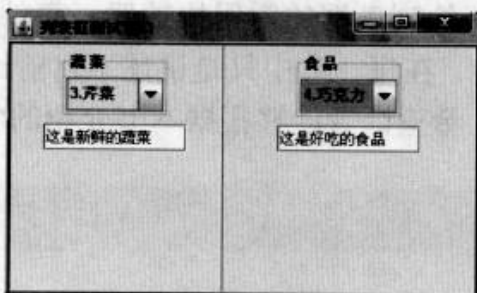


图 8.18 本章习题 4

第9章 如何使用进度条、时间、滑块和分隔条组件

在安装一个软件的时候经常会遇到进度条，进度条通常是用来显示一个操作的完成百分比，用户可以通过观察进度条得知目前操作进行到何种程度。时间组件用于在一段时间内依次做出程序员指定的操作。滑块可以显示主刻度标记和次刻度标记，刻度标记中间的值的个数由 `setMajorTickSpacing` 和 `setMinorTickSpacing` 来控制。分隔条就是平时在菜单项中看到的用来分隔不同类型菜单项的分隔线。时间组件主要是控制事件发生频率的组件。本章将针对以上 4 个组件配合实例给予详细讲解。

9.1 如何使用进度条组件 JProgressBar

进度条是用来表示一个操作的进程。用户可以通过进度条大概了解目前操作进度如何。下面将以表格的形式列出进度条的构造器，如表 9.1 所示。

表 9.1 JProgressBar 的构造器

JProgressBar 的构造器	说明
<code>JProgressBar()</code>	创建一个显示边框但不带进度字符串的水平进度条
<code>JProgressBar(BoundedRangeModel newModel)</code>	创建具有指定的保存进度条数据模型的水平进度条
<code>JProgressBar(int orient)</code>	创建具有指定方向（ <code>JProgressBar.VERTICAL</code> 、 <code>JProgressBar.HORIZONTAL</code> ）的进度条
<code>JProgressBar(int min,int max)</code>	创建具有指定最小值和最大值的水平进度条
<code>JProgressBar(int orient,int min,int max)</code>	创建具有指定方向、最小值和最大值的进度条

介绍了构造器后，下面将通过程序实例来学习如何创建进度条对象，其具体实例代码如下：

```
// 这段程序代码主要是为读者展示如何创建一个进度条组件
import javax.swing.*;
import java.awt.*;
public class test1
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("添加内容面板测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



```
jf.setVisible(true);
JPanel contentPane=new JPanel( );
jf.setContentPane(contentPane);
JProgressBar pb=new JProgressBar();           // 创建一个进度条对象
contentPane.add(pb);
}
```

上面程序代码的运行结果如图 9.1 所示。

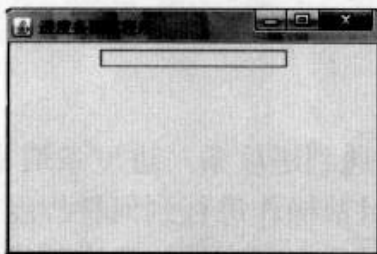


图 9.1 创建 JProgressBar 进度条 (a)

以上是使用最简单的构造器 `JProgressBar()` 来创建进度条。下面将使用另一个构造器 `JProgressBar(int orient)` 来创建进度条。其具体的实例代码如下：

```
// 这段程序代码主要是为读者展示使用 JProgressBar(int orient)构造器
// 创建一个带垂直方向的进度条组件
import javax.swing.*;
import java.awt.*;
public class test2
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("进度条测试程序");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JProgressBar pb=new JProgressBar(JProgressBar.VERTICAL);
        // 创建一个垂直方向的进度条对象
        contentPane.add(pb);
    }
}
```

上面程序的运行结果如图 9.2 所示。

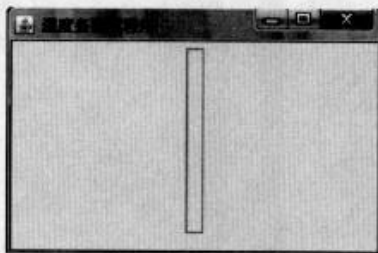


图 9.2 创建 JProgressBar 进度条 (b)

上面的程序用于创建一个竖直的进度条,那么如何才能使进度条在一定时间内动态表现其进度呢?要达到这种要求的话,进度条就必须配合另一个组件才有意义,那就是 Timer 组件,下一节将为读者详细讲述 Timer 组件的用法。

9.2 如何使用时间组件 Timer

利用 Timer 组件可以在一段时间内依次做出程序员指定的操作,这在动画的展示上非常有用。如果有用过如 ACDsee 类的看图软件,就可以发现这类软件都会提供一种功能,那就是自动换图功能,而且也可以设置换图时间间隔的长短。在 Java 中,Swing 的 Timer 组件就可以实现此功能,而且非常容易。下面先来观察 Timer 的构造器:Timer(int delay,ActionListener listener)用于创建一个每 delay 毫秒将通知其事件监听器的 Timer。

利用 Timer 组件会在根据所给予的 delay 时间内周期性地触发(ActionEvent 事件,如果要处理这个事件,必须实现 ActionListener 接口所定义的 actionPerformed()方法。要开始激活 Timer 组件可以使用 start()方法,要停止 Timer 组件可以使用 stop()方法,要从新激活 Timer 组件可以使用 restart()方法,若让 Timer 组件只触发一次(ActionEvent 事件,可利用 setRepeats(false)方法,将参数设为 false,若要设置 delay 时间则可利用 setDelay()方法。事实上,使用 Timer 组件,就是表示在程序后台是利用 Threads 在运行 Timer 的工作,因此当然也可以利用 Thread 的功能来自行制造出这样的效果。

根据进度条,再结合 Timer 组件,下面将为读者列举一个实例。该实例主要是演示一个进度条的动作事件。其程序代码如下所示:

```
// 这段程序代码主要是为读者展示一个正在运行的进度条组件,它结合了时间组件一起让进度条组件具有动画的效果
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;

public class test3 implements ActionListener,ChangeListener
{
    JFrame f = null;
    JProgressBar progressbar;
    JLabel label;
    Timer timer;
    JButton b;

    public test3()
    {
        f = new JFrame("progressbar Example");
        Container contentPane = f.getContentPane();
        label = new JLabel(" ",JLabel.CENTER);
        progressbar = new JProgressBar();
        progressbar.setOrientation(JProgressBar.HORIZONTAL);
        progressbar.setMinimum(0);
        progressbar.setMaximum(100);
        progressbar.setValue(0);
        progressbar.setStringPainted(true);
        progressbar.addChangeListener(this);
        progressbar.setPreferredSize(new Dimension(200,30));
        JPanel panel = new JPanel();

        // 创建一个进度条
        // 设置其方向为水平方向
        // 最小刻度 0
        // 最大刻度 100
        // 添加进度条变化事件
    }
}
```



```
b = new JButton("Start");
b.addActionListener(this); // 为按钮添加动作事件
panel.add(b);
timer = new Timer(50,this); // 创建一个事件组件对象
contentPane.add(panel,BorderLayout.NORTH);
contentPane.add(progressbar,BorderLayout.CENTER);
contentPane.add(label,BorderLayout.SOUTH);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    new test3();
}
public void actionPerformed(ActionEvent e)
{
    // 当单击按钮时, 计时开始
    if(e.getSource() == b)
    {
        timer.start();
    }
    // 当单击事件组件时, 进度条开始变化
    if(e.getSource() == timer)
    {
        int value = progressbar.getValue();
        if( value < 100)
        {
            value++; // 进度条往前运动
            progressbar.setValue(value);
        }
        else
        {
            timer.stop();
            progressbar.setValue(0);
        }
    }
}
public void stateChanged(ChangeEvent e1)
{
    int value = progressbar.getValue();
    // 当进度条运行时, 就将其进度显示在标签中
    if(e1.getSource() == progressbar)
    {
        label.setText("目前已完成进度: "+Integer.toString(value)+" %");
    }
}
}
```

上面程序代码的运行结果如图 9.3 所示。

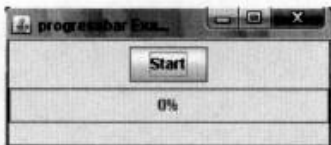


图 9.3 Timer 组件和 JProgressBar 组合

如果单击 Start 按钮，则进程开始进行，如图 9.4 所示。

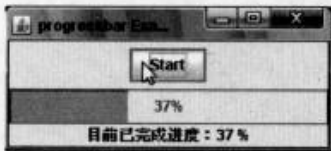


图 9.4 单击后的进度条

上面的程序实现了进度条在规定时间内动态变化的目的。在实际开发中，Timer 组件使用的频率较高。希望读者能够根据上述实例多多思考，从而真正掌握它的用法。

9.3 如何使用滑块组件 JSlider

滑块与进度条的功能很相似，只不过进度条是离散的选择项，而滑块是连续相同类型的值的设置，如选择 1~100 之间的任意值，滑块也可以称作调节条。滑块的构造器非常简单，如表 9.2 所示。

表 9.2 JSlider 构造器

JSlider 构造器	说明
JSlider()	创建一个范围在 0~100 之间并且初始值为 50 的水平滑块
JSlider(int orientation)	创建一个范围在 0 ~ 100 之间并且初始值为 50 的指定方向的滑块
JSlider(int min,int max,int value)	用指定的最小值、最大值和值创建一个水平滑块

平时见到的滑块是带有标尺的，也就是带有刻度的滑块，那么这些刻度是如何来的？其实，可以通过下面的方法来添加标尺刻度。

- setMajorTickSpacing(int n): 此方法用于设置主刻度标记的间隔。
- setMinorTickSpacing(int n): 此方法用于设置次刻度标记的间隔。

以上方法设置了标尺的刻度，如果要让刻度显示出来，就必须使用 setPaintTicks(true)方法。由于滑块的调整是连续的，可以通过 getValue()方法来得到滑块当前的值，在某些特殊的情况下，程序可能需要整数值，但在滑动的时候，是无法保证所取到的值是整数，所以可以通过强制滑块滑到离自己最近的整数标尺处，而该功能由 setSnapToTicks(true)方法实现。

以上方法只能让刻度显示出来，但是无法知道刻度是多少，方法 setPaintLabels(true)将会提供如何添加数字标识。

下面将通过实例来讲解如何创建滑块。其程序代码如下：

```
// 这段程序代码主要是为读者展示创建滑块的方法
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
```



```
import java.awt.event.*;
import javax.swing.event.*;
public class test4
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("如何使用滑块");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        JSlider s=new JSlider(0,100,0);           // 创建一个滑块对象
        contentPane.add(s);
        s.setMajorTickSpacing(20);               // 设置主刻度
        s.setMinorTickSpacing(5);               // 设置次刻度
        s.setPaintTicks(true);                   // 让刻度显现出来
        s.setSnapToTicks(true);                 // 让滑块滑到附近的整数处
        s.setPaintLabels(true);                 // 让刻度上的数字显示出来
    }
}
```

上面程序代码的运行结果如图 9.5 所示。

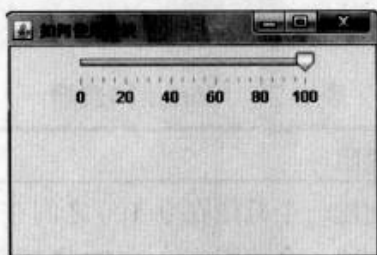


图 9.5 如何创建滑块

以上为读者讲述了如何创建滑块，下面将针对滑块的事件处理给予讨论。

当拖动滑块改变其值的时候会激发 `ChangeEvent` 事件，而监听该事件的监听器需要实现 `ChangeListener`，该接口只有一个方法 `stateChanged`。下面将通过实例来讲解滑块在一般开发中的事件处理过程。其程序代码如下：

```
import javax.swing.*;
import java.awt.*;
import javax.swing.event.*;
public class test5
{
    static final int WIDTH=300;
    static final int HEIGHT=200;
    public static void main(String[] args)
    {
        JFrame jf=new JFrame("如何使用滑块");
        jf.setSize(WIDTH,HEIGHT);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
        JPanel contentPane=new JPanel( );
        jf.setContentPane(contentPane);
        contentPane.setLayout(new BorderLayout());
```



```

final JSlider s=new JSlider(0,100,0);           // 创建一个滑块对象
contentPane.add(s,"South");
s.setMajorTickSpacing(20);                     // 设置主刻度
s.setMinorTickSpacing(5);                     // 设置次刻度
s.setPaintTicks(true);                         // 让刻度显示出来
s.setSnapToTicks(true);                       // 让滑块能够滑到附近整数刻度上
s.setPaintLabels(true);                       // 让刻度数字显示出来
final JLabel l=new JLabel("目前刻度: "+s.getValue());
contentPane.add(l,"North");
// 当拖动滑块时产生的事件
s.addChangeListener(new ChangeListener()
{
    public void stateChanged(ChangeEvent e)
    {
        // 当滑动滑块时就会将刻度数显示在标签上
        if ((JSlider)e.getSource() == s)
            l.setText("目前刻度: "+s.getValue());
    }
});
}
}

```

上面程序代码的运行结果如图 9.6 所示。

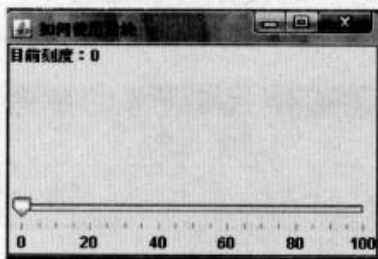


图 9.6 如何处理滑块的事件处理

当拖动滑块时，上面标签中的数字也会紧随着改变，如图 9.7 所示。

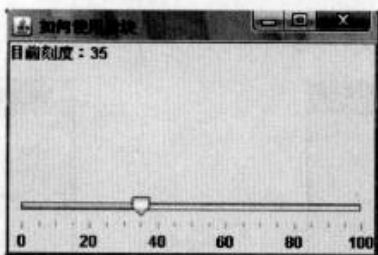


图 9.7 拖动后的滑块

上述实例很简单，但是它说明了一个滑块需要处理事件时的基本处理方式。希望读者能够自行编写实例练习如何处理这一类事件。

9.4 如何使用分隔条组件 JSeparator

JSeparator 为实现分隔线提供了一个通用组件。在实际开发中，通常用作菜单项之间的分隔符，以便将不同类型的菜单项分成几个逻辑组。添加一个分隔符不是直接使用 JSeparator，而是使用 addSeparator 方法来实现。

在除了菜单项中的分隔符使用 `addSeparator` 方法创建和添加外，在很多其他的 GUI 图形设计中，都是使用分隔符的构造器来创建分隔符。其构造器如下所示。

- `JSeparator()`: 创建一个新的水平分隔符。
- `JSeparator(int orientation)`: 创建一个具有指定水平或者垂直方向的分隔符。

由于分隔符比较简单，所以在这里不再给出实例。

9.5 本章小结

本章主要是针对分隔符、时间组件、滑块、分隔条 4 种组件结合实例进行了详细讲解和分析，并且在讲解每一个控件的时候，都会提及在一般实际开发中这些组件的注意事项。每个实例都很贴近实际开发，希望读者能够根据本章的实例进行深入学习。

9.6 本章习题

1. 设计一个程序，在程序中有两个按钮组件和一个进度条组件，当单击“开始”按钮时，进度条在运行，单击“停止”按钮时，进度条停止运行。

要求：其效果如图 9.8 所示。

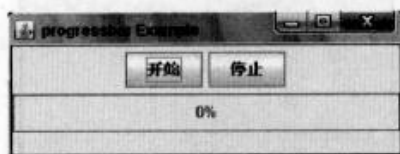
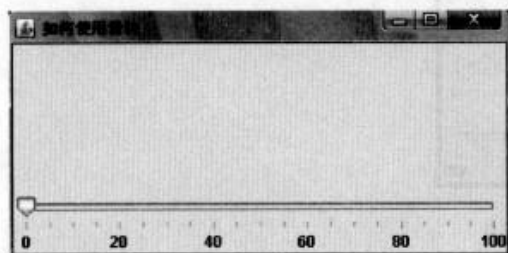


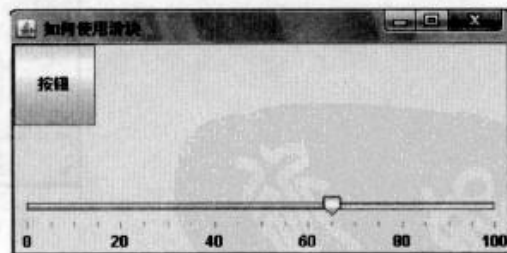
图 9.8 本章习题 1

2. 设计一个程序，在程序中有一个滑块组件和一个按钮组件，当滑块滑动时，按钮组件大小会随之发生变化。

要求：程序运行的效果如图 9.9 所示。



(a)



(b)

图 9.9 本章习题 2

3. 设计一个程序，利用进度条组件在程序中设计一个程序安装界面。

要求：

(1) 其效果如图 9.10 所示。



图 9.10 本章习题 3 (a)

(2) 当单击图中按钮时, 会弹出一个安装进度的界面, 如图 9.11 所示。



图 9.11 本章习题 3 (b)

4. 设计一个程序, 利用进度条组件在程序中设计一个程序卸载界面。

要求: 其运行后的界面如图 9.10 所示, 当单击按钮组件后, 出现的进度条组件会反向运行, 如图 9.12 所示。



图 9.12 本章习题 4

第10章 如何使用选取器组件

本章主要为读者介绍有关选取器的知识。所谓的选取器，通俗地说就是在一系列数据中选择自己需要的数据。在 Swing 类库中，选取器有两种，一种是文件选取器 JFileChooser，另一种是颜色选取器 JColorChooser。文件选取器 JFileChooser 就是大家平时在操作系统中见到的保存和打开窗口。颜色选取器 JColorChooser 则是大家平时在画图软件或者是 Photoshop 软件中用到的颜色选择组件。在这一章中将会以理论为基础、以程序分析为实践，为读者重点讲述它们在实际开发中的用法。

10.1 如何使用文件选取器 JFileChooser

如果一个文本编辑器上打了一段文字，可能希望将此段文字存储起来，以供日后使用，此时系统应当提供一个存储文件的对话框，将此段文字存储到一个自定义文件名或内定的文件夹中。同样，要打开某个文件时，系统也应当提供打开文件的对话框，让用户在众多的文件中选择欲打开的文件。

在 Java 中这些操作都可以由文件选取器 JFileChooser 组件来完成。这个组件提供了打开文件和保存窗口的功能、显示特定类型文件图标的功能，也能针对某些文件类型进行过滤操作。如果系统需要对某些文件或文件夹进行操作，JFileChooser 组件可以让开发人员轻松地开发出漂亮的用户界面。需要注意的是 JFileChooser 本身不提供读文件或存盘的功能，这些功能必须自行实现。事实上，JFileChooser 本身只是一个对话框模型，它也是依附在 JDialog 的结构上，JFileChooser 的构造器的说明如表 10.1 所示。

表 10.1 JFileChooser 构造器

JFileChooser 构造器	说明
JFileChooser()	建立一个 JFileChooser 对象，默认的文件对话框路径是用户的 home 目录。例如在 Windows 2000 中的 Administrator 的目录是 C:\DocumentsandSettings\Administrator
JfileChooser(File currentDirectory)	建立一个 JFileChooser 对象，并以 File 所在位置为文件对话框的打开路径
JFileChooser(File currentDirectory, FileSystemView fsv)	建立一个 JFileChooser 对象，以 File 所在位置为文件对话框的打开路径并设置图标查看方式
JfileChooser(FileSystemView fsv)	建立一个 JFileChooser 对象并设置文件图标查看方式

JFileChooser 构造器	说明
JFileChooser(String currentDirectoryPath)	建立一个 JFileChooser 对象并设置文件对话框的打开路径
JFileChooser(String currentDirectoryPath, FileSystemView fsv)	建立一个 JFileChooser 对象并设置文件对话框的打开路径与文件图标查看方式

下面将为读者详细介绍如何创建文件选取器 JFileChooser 组件。

10.1.1 如何创建 JFileChooser 组件

文件选择器 JFileChooser 中的 showOpenDialog()或 showSaveDialog()方法用来打开文件对话框和保存文件对话框，两个方法在用户按下按钮或关闭对话框时都会返回一个整数值，这个整数值类型有 3 种，如表 10.2 所示。

表 10.2 文件选取器返回的几个整数值

整数值	说明
JFileChooser.CANCEL_OPTION	表示用户单击“取消”按钮
JFileChooser.APPROVE_OPTION	表示用户单击“确定”按钮
JFileChooser.ERROR_OPEION	表示有错误产生或是对话框不正常关闭

利用这 3 个整数值就能判断用户到底在对话框中进行了什么操作，并加以处理。下面给出一个实例，该实例主要是创建一个文件保存对话框并且处理其事件，以及创建一个打开对话框且处理其事件，通过这个实例可使读者熟悉 JFileChooser 组件的具体用法，实例代码如下：

```
// 这段程序代码主要是创建一个文件保存对话框以及打开对话框
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

class test1 implements ActionListener           // 这个类实现了动作监听类
{
    JFrame f = null;
    JLabel label = null;
    JTextArea textarea = null;
    JFileChooser fileChooser = null;
    public test1()
    {
        f = new JFrame("FileChooser Example");
        Container contentPane = f.getContentPane();
        textarea = new JTextArea();
        JScrollPane scrollPane = new JScrollPane(textarea);
        scrollPane.setPreferredSize(new Dimension(350,300));
        JPanel panel = new JPanel();
        JButton b1 = new JButton("新建文件");
        b1.addActionListener(this);
        JButton b2 = new JButton("存储文件");
        b2.addActionListener(this);
        panel.add(b1);
```



```

panel.add(b2);
label = new JLabel(" ", JLabel.CENTER);
// 建立一个 FileChooser 对象, 并指定 D 的目录为默认文件对话框路径
fileChooser = new JFileChooser("D:\\");
contentPane.add(label, BorderLayout.NORTH);
contentPane.add(scrollPane, BorderLayout.CENTER);
contentPane.add(panel, BorderLayout.SOUTH);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
...
}

```

以上代码创建了一个程序的主界面, 但是如果要实现“新建文件”按钮组件的动作功能, 就必须要实现以下的代码:

```

public static void main(String[] args)
{
    new test1();
}
public void actionPerformed(ActionEvent e)
{
    File file = null;
    int result;
    /* 当用户按下"新建文件"按钮时, JFileChooser 的 showOpenDialog()方法会输出文件对话框
    * 并利用 setApproveButtonText()方法取代按钮上 Open 文字
    * 以 setDialogTitle()方法设置打开文件对话框 Title 名称, 当选择完后, 会将选择结果存到 result 变量中
    */
    if (e.getActionCommand().equals("新建文件"))
    {
        fileChooser.setApproveButtonText("确定");
        fileChooser.setDialogTitle("打开文件");
        result = fileChooser.showOpenDialog(f);
        textarea.setText("");
        /* 当用户按下打开文件对话框的"确定"按钮后, 就可以利用 getSelectedFile()方法取得文件对象
        * 若是用户按下打开文件对话框的 Cancel 按钮, 则将在 label 上显示"你没有选择任何文件"字样
        */
        if (result == JFileChooser.APPROVE_OPTION)
        {
            file = fileChooser.getSelectedFile();
            label.setText("您选择打开的文件名称为: "+file.getName());
        }
        else if (result == JFileChooser.CANCEL_OPTION)
        {
            label.setText("您没有选择任何文件");
        }
    }
}

```



```
...
}
}
```

以上代码用于打开一个“打开”对话框，但是当选择了文件后，如何才能将选择的文本文件显示在文本组件中呢？那就要使用到输入输出流的知识，其代码如下：

```
FileInputStream fileInStream = null;
if(file != null)
{
    try
    {
        // 利用 FileInputStream 将文件内容放入此数据流中以便读取
        fileInStream = new FileInputStream(file);
    }
    catch(FileNotFoundException fe)
    {
        label.setText("File Not Found");
        return;
    }
    int readbyte;
    try
    {
        /* 以 read()方法读取 FileInputStream 对象内容，当返回值为-1 时代表读完此数据流
        * 将所读到的字符显示在 textarea 中
        */
        while( (readbyte = fileInStream.read()) != -1)
        {
            textarea.append(String.valueOf((char)readbyte));
        }
    }
    catch(IOException ioe)
    {
        label.setText("读取文件错误");
    }
    Finally
    {
        // 回收 FileInputStream 对象，避免资源的浪费
        Try
        {
            if(fileInStream != null)
                fileInStream.close();
        }
        catch(IOException ioe2){}
    }
}
```

如果要实现“存储文件”按钮组件功能，就必须实现下面的代码：

```
if(e.getActionCommand().equals("存储文件"))// 实现写入文件的功能
{
    result = fileChooser.showSaveDialog(f);
```



```
file = null;
String fileName;
// 当用户没有选择文件, 而是自己键入文件名称时, 系统会自动以此文件名建立新文件
if (result == JFileChooser.APPROVE_OPTION)
{
    file = fileChooser.getSelectedFile();
    label.setText("您选择存储的文件名称为: "+file.getName());
}
else if(result == JFileChooser.CANCEL_OPTION)
{
    label.setText("您没有选择任何文件");
}
}
```

同样, 保存文件也需要使用到输入输出流的知识, 其代码如下所示:

```
FileOutputStream fileOutputStream = null;
/* 使用 FileOutputStream 写入文件, 在这个范例中写入文件的方式是将之前内容清除并重新写入
 * 若想把新增的内容加在原有的文件内容后, 可以使用 FileOutputStream(String name, Boolean append)构造函数
 */
if(file != null)
{
    Try
    {
        fileOutputStream = new FileOutputStream(file);
    }
    catch(FileNotFoundException fe)
    {
        label.setText("File Not Found");
        return;
    }
    String content = textarea.getText();
    Try
    {
        fileOutputStream.write(content.getBytes());
    }
    catch(IOException ioe)
    {
        label.setText("写入文件错误");
    }
    Finally
    {
        try
        {
            if(fileOutputStream != null)
                fileOutputStream.close();
        }
        catch(IOException ioe2){}
    }
}
```

以上程序代码的运行结果如图 10.1 所示。



图 10.1 创建 JFileChooser 组件

当单击“新建文件”按钮后，会出现如图 10.2 所示对话框。



图 10.2 “打开文件”对话框

当单击“存储文件”按钮时，会出现如图 10.3 所示对话框。



图 10.3 “打开文件”对话框

上面的程序看上去很复杂，其实很简单，其流程如图 10.4 所示。

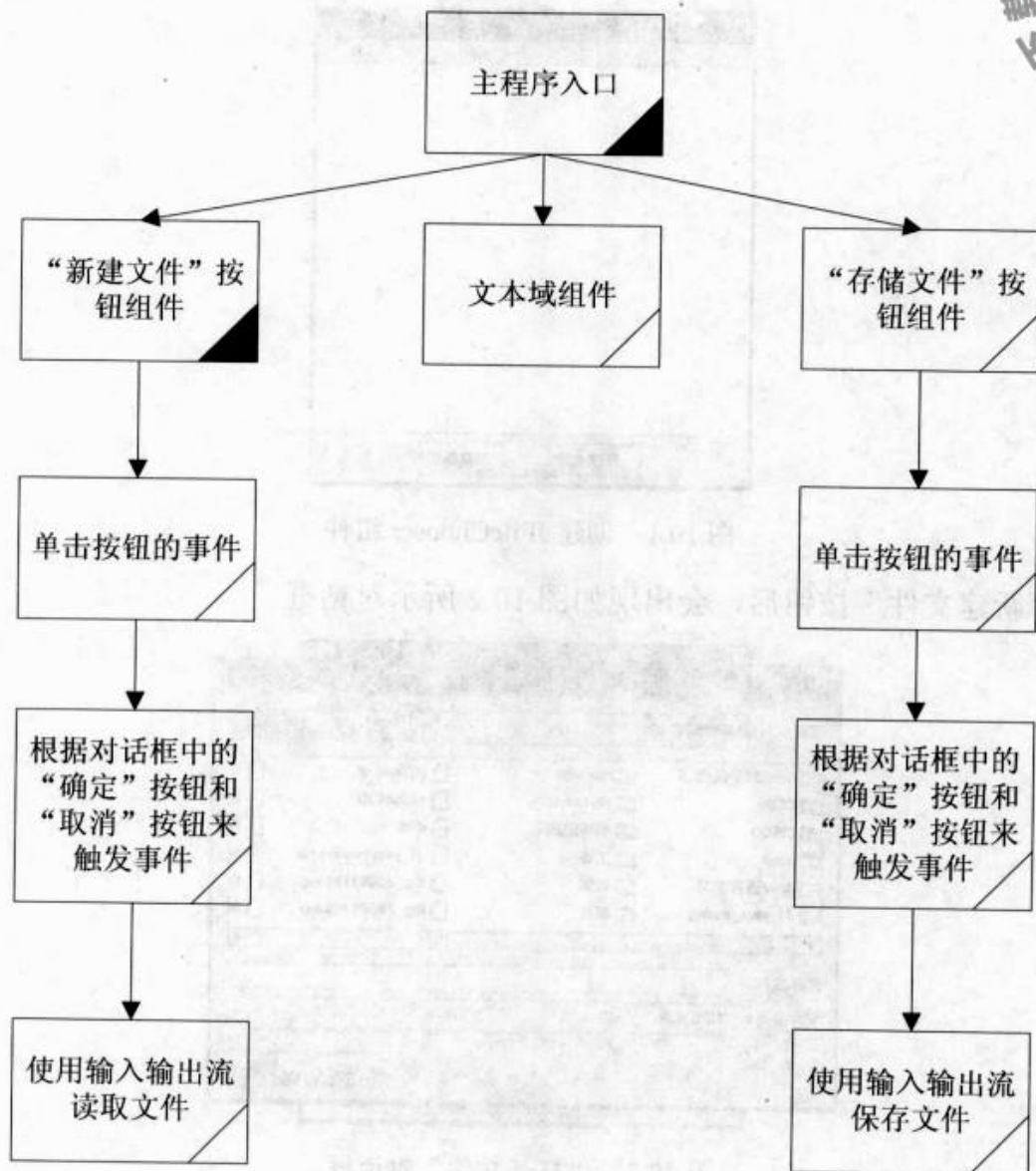


图 10.4 流程图

10.1.2 如何创建 JFileChooser 对话框

如果专为特定文件类型设计一套软件时，为了让用户打开文件时存盘方便，通常会在文件对话框中过滤掉无关的文件类型，使用户快速选择出想要的文件数据，例如在 Word 软件中，当选择“文件另存为”命令时，所出现的文件对话框将会以“.doc”扩展名为默认的文件存储类型。

如果想在 Java 的文件对话框中实现这样的功能，必须实现 `FileFilter` 抽象类。此抽象类里定义了两个空的方法，分别是 `accept` 与 `getDescription()`。当目录里的文件与设置的文件类型相符时，`accept()`方法就会返回 `true`，并将此文件显示在文件对话框中。而 `getDescription()`方法则是对此文件类型的描述，可由程序设计者自定义，如“*.java”等。要设置选择文件类型对话框可以利用 `JFileChooser` 的 `addChoosableFileFilter()`方法或是 `setFileFilter()`方法。根据以上讲述的内容将给出一个实例，程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建一个具有可选择文件类型的对话框
import java.awt.*;
import javax.swing.*;
```



```

import java.awt.event.*;
// 由于在程序中使用到 File 与 FileFilter 对象, 因此要使用 import File 与 FileFilter 这两个类
import java.io.File;
import javax.swing.filechooser.FileFilter;
public class test2 implements ActionListener
{
    JFrame f=null;
    JLabel label=null;
    JFileChooser fileChooser=null;
    public test2()
    {
        f=new JFrame("FileFilterDemo");
        Container contentPane=f.getContentPane();
        JButton b=new JButton("打开文件");
        b.addActionListener(this);
        label=new JLabel(" ",JLabel.CENTER);
        label.setPreferredSize(new Dimension(150,30));
        contentPane.add(label,BorderLayout.CENTER);
        contentPane.add(b,BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        new test2();
    }
    public void actionPerformed(ActionEvent e)                // 处理用户按下"打开文件"按钮事件
    {
        fileChooser=new JFileChooser("C:\\winnt");                // 以 c:\\winnt 为打开文件对话框的默认路径
        fileChooser.addChoosableFileFilter(new JAVAFileFilter("class"));
        // 利用 addChoosableFileFilter()方法加入欲过滤的文件类型
        // 使用 addChoosableFileFilter()可以加入多种文件类型
        // 若只需要过滤出一种文件类型, 可使用 setFileFilter()方法
        fileChooser.addChoosableFileFilter(new JAVAFileFilter("java"));
        int result=fileChooser.showOpenDialog(f);
        // 如果单击"确定"按钮的话, 则可以打开现有的文件
        if (result==JFileChooser.APPROVE_OPTION)
        {
            File file=fileChooser.getSelectedFile();
            label.setText("你选择了:"+file.getName()+"文件");
        }
        else if (result==fileChooser.CANCEL_OPTION)
        {
            // 如果单击"取消"按钮的话, 那么就会在标签中显示"你没有选取文件"
            label.setText("你没有选取文件");
        }
    }
}

```


如果要选择固定的文件类型，就要创建一个类，这个类继承 `FileFilter` 抽象类，实现其中的方法，从而实现文件类型过滤的功能。其代码如下所示：

```
// 以 JAVAFileFilter 类继承 FileFilter 抽象类，并实现 accept()与 getDescription()方法
class JAVAFileFilter extends FileFilter
{
    String ext;
    public JAVAFileFilter(String ext)
    {
        this.ext=ext;
    }
    public boolean accept(File file)
    // 在 accept()方法中，当程序返回的是一个目录而不是文件时，返回 true 值，表示将此目录显示出来
    {
        if (file.isDirectory())
        {
            return true;
        }
        String fileName=file.getName();
        int index=fileName.lastIndexOf('.');
        if (index>0 && index<fileName.length()-1)
        {
            String extension=fileName.substring(index+1).toLowerCase();// 表示文件名称不为".xxx"或"xxx."类型
            if (extension.equals(ext))
            // 若返回的文件扩展名等于所设置的扩展名，则返回 true，表示将此文件显示出来，否则返回 false
            {
                return true;
            }
        }
        return false;
    }
    public String getDescription()
    {
        // 实现 getDescription()方法，返回描述文件的说明字符串
        if (ext.equals("java"))
            return "JAVA Source File(*.java)";
        if (ext.equals("class"))
            return "JAVA Class File(*.class)";
        return "";
    }
}
```

上面程序代码的运行结果如图 10.5 所示。



图 10.5 建立可选择文件类型的 JFileChooser 对话框

当单击“打开文件”按钮时，就会出现如图 10.6 所示对话框。

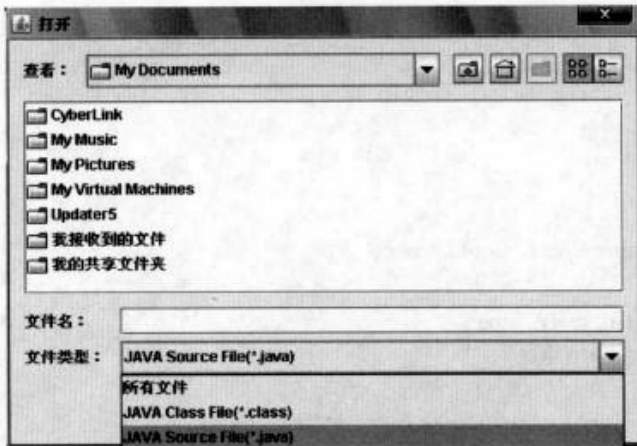


图 10.6 “打开”对话框

以上就是文件选取器的相关知识，还有一些其他的特殊情况，希望读者能够查阅相关的资料自行学习。

10.2 如何使用颜色选取器 JColorChooser

ColorChooser 可以让用户选择自己想要的颜色并更改某个组件的颜色，例如在小画家中可以在画板上绘制图案，并选择各式各样的颜色来加以装饰；至于颜色的选择，可以在小画家中找到颜色选择对话框。颜色选择对话框可以通过使用颜色选取器来创建。

颜色选取器 JColorChooser 的构造器的说明如表 10.3 所示。

表 10.3 颜色选取器 JColorChooser 的构造器

颜色选取器的构造器	说明
JColorChooser()	建立一个 JColorChooer 对象，默认颜色为白色
JColorChooser(Color initialColor)	建立一个 JColorChooer 对象，并设置初始颜色
JColorChooser(ColorSelectionModel modal)	以 ColorSelectionModel 构造 JColorChooser 对象

下面将通过实例向读者介绍如何使用、如何创建颜色选择框，其实例代码如下所示：

```
/* 这段程序代码主要是为读者展示如何创建一个颜色选取框，在程序中主要是创建了 5 个普通按钮组件
 * 第一个按钮组件用来获得在颜色选取框中选择的颜色，后面 4 个普通按钮组件主要是显示所选颜色的参数
 */
import java.awt.Dimension;
import javax.swing.JColorChooser;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;
import java.awt.FlowLayout;
public class test3 implements ActionListener
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame ("JColorChooserDemo");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
```



```

        MyPanel panel = new MyPanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
class MyPanel extends JPanel implements ActionListener
{
    private JButton button, rgb, red, green, blue;
    private Color color = new Color(0, 0, 0);
    public MyPanel()
    {
        button = new JButton("Get Color");
        rgb = new JButton("RGB: ");
        red = new JButton("Red: ");
        green = new JButton("Green: ");
        blue = new JButton("Blue: ");
        button.addActionListener(this);
        setPreferredSize(new Dimension(550, 250));
        setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        setBackground(color);
        add(button);
        add(rgb);
        add(red);
        add(green);
        add(blue);
    }
    public void actionPerformed(ActionEvent e)
    {
        color = JColorChooser.showDialog(this, "Choose Color", color);    // 创建一个颜色选取器
        setBackground(color);
        button.setText("Get again");
        rgb.setText("RGB: " + color.getRGB());    // 通过 get 方法来获取颜色库中的颜色
        red.setText("Red: " + color.getRed());
        green.setText("Green: " + color.getGreen());
        blue.setText("Blue: " + color.getBlue());
    }
}

```

上面程序代码的运行结果如图 10.7 所示。

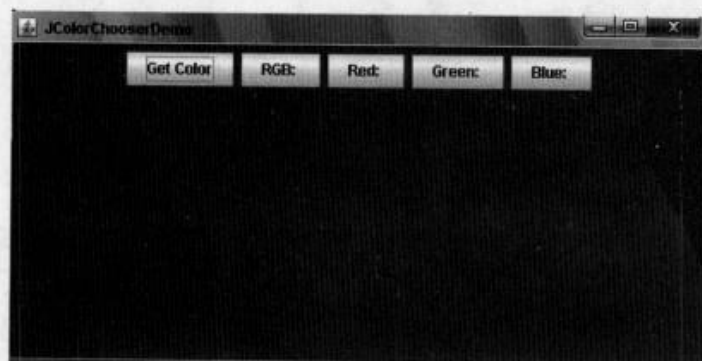


图 10.7 颜色选取器的使用 (a)

当单击 Get Color 按钮时，会弹出颜色选取框，如图 10.8 所示。

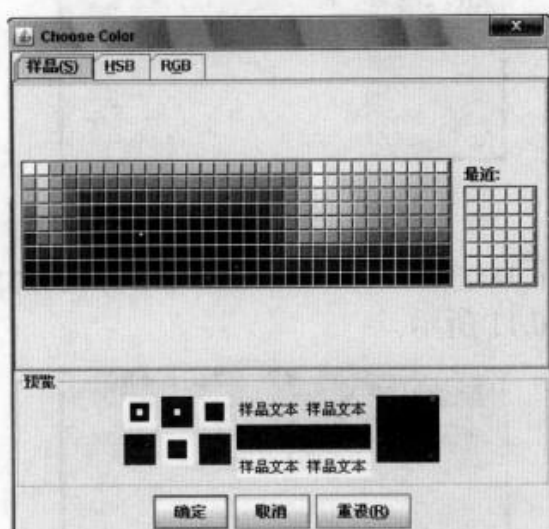


图 10.8 颜色选取器的使用 (b)

当单击“确定”按钮后，界面上会出现所选颜色，如图 10.9 所示。



图 10.9 颜色选取器的使用 (c)

在实际开发中，颜色选取器的使用不是很频繁，只是一些图形类的软件会使用到它。限于篇幅的原因，这里不再赘述。

10.3 本章小结

本章主要是向读者讲述了使用文件选取器和颜色选取器的基本方法，并且列举了大量的实例和例图。其实，有关这方面的知识还有很多，有兴趣的读者可以查阅相关的资料自行练习。

10.4 本章习题

1. 设计一个程序，程序是一个菜单，通过菜单中的“打开”命令来新建一个文本文档。
要求：
(1) 其程序运行效果如图 10.10 所示。



图 10.10 本章习题 1 (a)

(2) 选择“文件”|“打开 txt 文档”命令，选择需要的“.txt”文档，文档内容就会出现在文本组件中，其效果如图 10.11 所示。

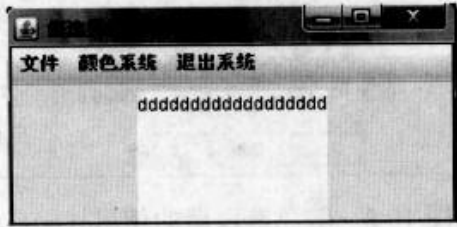


图 10.11 本章习题 1 (b)

2. 设计一个程序，根据上面习题中所创建的 Word 文档，利用“打开”命令打开 Word 文档。
3. 设计一个程序，根据菜单项中的“保存”命令来保存新建的 Word 文档。
4. 设计一个程序，在程序中将文档的背景颜色设置成红色。

第11章 如何使用文本组件

在实际的图形化程序界面开发中，开发人员经常会给用户留有一些可以输入信息的组件，这些组件被称为文本组件。在 Swing 图形化开发中，将文本组件分成了普通文本组件和文本区组件两种。它们之间的区别在于，普通文本组件只可以接收用户输入的单行文本，而文本区组件可以接收用户输入的多行文本。在普通文本组件中，还分为文本域组件、密码文本域组件、带格式的文本域组件三种。本章将会针对以上的知识点给予详细讨论。

11.1 文本组件概述

在 Swing 图形开发库中，与文字输入有关的组件分别是 JTextField、JPasswordField、JFormattedTextField、JTextArea、JEditorPane 与 JTextPane，其中 JTextField 与 JPasswordField 为单行的文本编辑器；JTextArea 为多行的文本编辑器；JEditorPane 为可显示多种文件格式的组件；JTextPane 可设置文件的各种样式。这些组件都继承自 JTextComponent 类，它们之间的关系如图 11.1 所示。

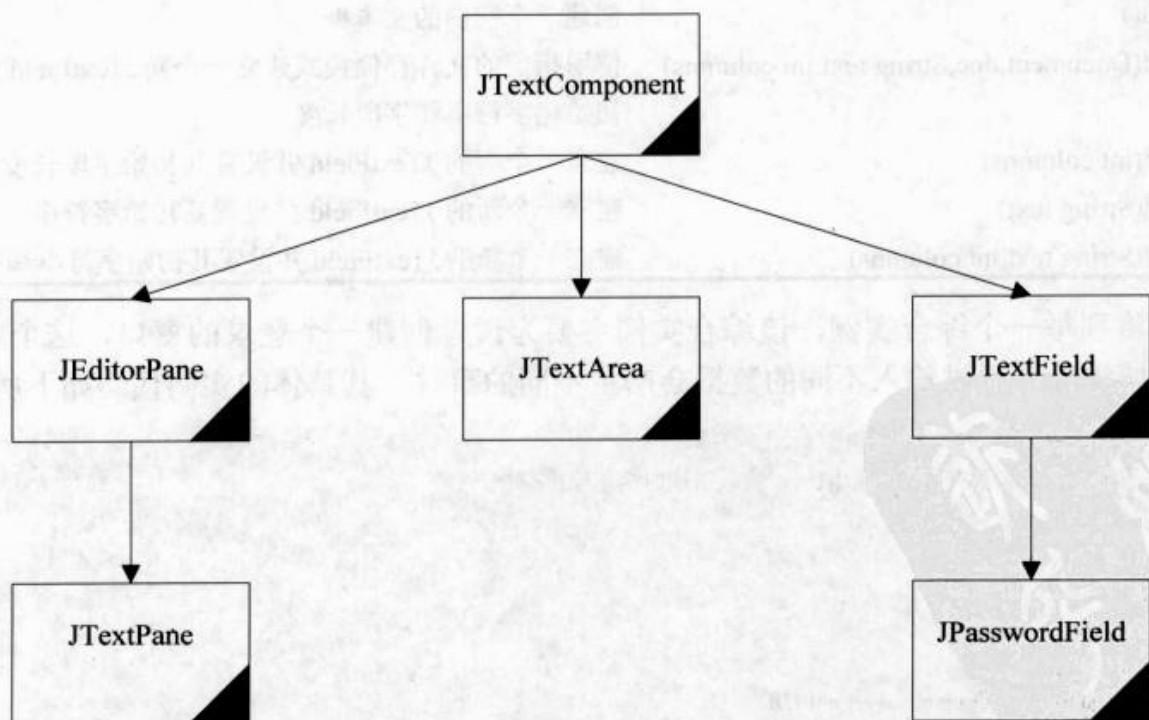


图 11.1 文本组件的继承关系图

JTextComponent 提供了相当多的实用方法，可使处理输入组件更为方便，例如 copy()、paste()、cut()、getText()、setText()等。另外，设置是否可编辑（setEditable()、setSelectionEnd()、setSelectionStart()）、设置或取得光标位置（getCaretPosition()、setCaretPosition()）等，这些比较常用的方法都可以在 JTextComponent 类中找到。

11.2 如何使用普通文本组件

本节主要讨论普通文本组件的使用，普通文本组件允许用户只能输入单行数据信息的文本组件，其中包括 JTextField、JPasswordField、JFormattedTextField。在实际开发中，JTextField 文本组件一般是用来输入一些简短的信息，如用户名等。JPasswordField 文本组件用来输入密码信息，如密码框等。JFormattedTextField 文本组件用来按照固定的格式输入文本信息，如果创建对象的时候规定输入格式是整数类型，那么当输入其他类型的数据后，当焦点离开文本框时，文本框中的数据会自动变成整数类型的数据，具体的使用方法将会在下面的内容中给予详细描述。

11.2.1 如何使用 JTextField

JTextField 继承自 JTextComponent 类，因此它也可以使用 JTextComponent 抽象类里的方法，如 copy()、paste()、setText()、isEditable()等。JTextField 是一个单行的输入组件，下面将以表格的形式为读者列出 JTextField 构造器，如表 11.1 所示。

表 11.1 JTextField 构造器

JTextField 构造器	说明
JTextField()	创建一个空白的文本框
JTextField(Document doc,String text,int columns)	使用指定的文件存储模式建立一个新 JTextField 并设置其初始化字符串和字段长度
JTextField(int columns)	建立一个新的 JTextField 并设置其初始字段长度
JTextField(String text)	建立一个新的 JTextField 并设置其初始字符串
JTextField(String text,int columns)	建立一个新的 JTextField 并设置其初始字符串和字段长度

下面将列举一个综合实例，该综合实例主要为读者创建一个登录的窗口，这个窗口包括两个文本域组件，通过输入不同的数据会产生不同的事件，其具体的实例代码如下所示：

```
// 这段程序代码主要创建一个登录窗口
// 输入正确的用户名和密码后窗口消失，输入错误时会清空密码框
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class test extends JPanel
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=300;
    static final int HEIGHT=150;
    JFrame loginframe;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
```



```

{
    constraints.gridx=x;
    constraints.gridy=y;
    constraints.gridwidth=w;
    constraints.gridheight=h;
    add(c,constraints);
}
test1()
{
    loginframe=new JFrame("一个综合实例");
    loginframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    GridBagLayout lay=new GridBagLayout();
    setLayout(lay);
    loginframe.add(this, BorderLayout.WEST);
    loginframe.setSize(WIDTH,HEIGHT);
    Toolkit kit=Toolkit.getDefaultToolkit();
    Dimension screenSize=kit.getScreenSize();
    int width=screenSize.width;
    int height=screenSize.height;
    int x=(width-WIDTH)/2;
    int y=(height-HEIGHT)/2;
    loginframe.setLocation(x,y);
    JButton ok=new JButton("登录");
    JButton cancel=new JButton("放弃");
    JLabel title=new JLabel("登录窗口");
    JLabel name=new JLabel("用户名");
    JLabel password=new JLabel("密 码");
    final JTextField nameinput=new JTextField(15);
    final JTextField passwordinput=new JTextField(15);
    GridBagConstraints constraints=new GridBagConstraints();
    constraints.fill=GridBagConstraints.NONE;
    constraints.anchor=GridBagConstraints.EAST;
    constraints.weightx=3;
    constraints.weighty=4;
    add(title,constraints,0,0,2,1);
    add(name,constraints,0,1,1,1);
    add(password,constraints,0,2,1,1);
    add(nameinput,constraints,2,1,1,1);
    add(passwordinput,constraints,2,2,1,1);
    add(ok,constraints,0,3,1,1);
    add(cancel,constraints,2,3,1,1);
    loginframe.setResizable(false);
    loginframe.setVisible(true);
    ok.addActionListener(new ActionListener()
    { // 处理登录按钮组件的动作事件
        public void actionPerformed(ActionEvent Event)
        {
            String nametext=nameinput.getText();
            String passwordtext=passwordinput.getText();
            String str=new String(passwordtext);
            boolean x=(nametext.equals("starsong")); // 在此设置密码和用户名
            boolean y=(str.equals("750720"));
            boolean z=(x&& y); // 当输入正确的密码和用户名后，主窗口就会消失
            if (z==true)
            {

```



```
        loginframe.dispose();
    }
    else if(z==false)
    {
        nameinput.setText("");
        // 当输入不正确的用户名和密码时，系统会将文本框内容清空
        passwordinput.setText("");
    }
    }
    });
}
}

public class test1
{
    public static void main(String[] args)
    {
        test log=new test();
    }
}
```

上面程序代码的运行结果如图 11.2 所示。

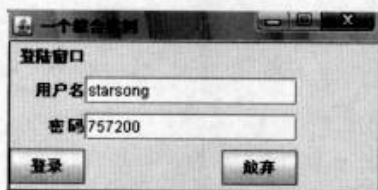


图 11.2 一个带文本组件的综合实例 (a)

当输入正确的用户名和密码后，窗口就会消失，输入不正确时就会出现如图 11.3 所示对话框。



图 11.3 一个带文本组件的综合实例 (b)

以上实例是一个简单的综合实例，其中包括前面讲解的几种组件的用法以及布局管理器和事件监听处理的知识。

下面将介绍另一种创建文本域组件的构造器，即 `JTextField(Document doc,String text,int columns)`，使用指定的文件存储模式建立一个新的 `JTextField` 并设置其初始化字符串和字段长度。

`Document` 接口的主要功能是定义一些方法，在使用所有与 `Text` 相关的组件时，能够将输入文字的内容加以结构化或规格化，具体的知识这里不再赘述，有兴趣的读者可以自行查阅相关资料，由以上的构造器可以知道，必须实现 `Document` 接口中的所有方法，才能利用 `Document` 构造出 `JTextField`，这种做法很麻烦，下面将为读者介绍一个比较简单的创建方法。

利用继承 `AbstractListModel` 的抽象类来构造 `JList`，由于抽象类已经实现了许多接口的方法，所以当继承这个抽象类后便不需要实现这些方法。同样的，Java 在这里也提供了一个

AbstractDocument 的抽象类, 以供开发人员使用, 从而为开发人员带来许多方便。

在这里并不是要使用 AbstractDocument, 因为 Java 在这部分已经提供了一个实体类: PlainDocument。这个实体类继承自 AbstractDocument, 也就是具备了所有 AbstractDocument 的方法, 所以只要直接继承 PlainDocument 实体类就能利用 Document 来构造 JTextField。

下面将通过使用上面介绍的方法来创建文本域组件, 其具体实例代码如下:

```
/* 这段程序代码主要是使用 PlainDocument 作为构造器 JTextField(Document doc,String text,int columns)
 * 的一个参数来创建文本域组件
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
public class test2
{
    public static void main(String args[])
    {
        JFrame f = new JFrame("JTextField3");
        Container contentPane = f.getContentPane();
        contentPane.setLayout(new BorderLayout());
        JPanel p1 = new JPanel();
        p1.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.anchor = GridBagConstraints.WEST; // 设定 Layout 的位置
        gbc.insets = new Insets(2,2,2,2); // 设定与边界的距离 (上、左、下、右)
        p1.setBorder(BorderFactory.createTitledBorder("学生的基本数据"));
        JLabel l1 = new JLabel("学生姓名: ");
        JLabel l2 = new JLabel("学生性别: ");
        JLabel l3 = new JLabel("学生身高: ");
        JLabel l4 = new JLabel("学生体重: ");
        // 创建 JTextField3_FixedLengthDocument()对象, 字符串为空, 指定长度的文本域组件
        JTextField t1 = new JTextField(new JTextField3_FixedLengthDocument(10),"",10);
        JTextField t2 = new JTextField(new JTextField3_FixedLengthDocument(1),"",1);
        JTextField t3 = new JTextField(new JTextField3_FixedLengthDocument(5),"",5);
        JTextField t4 = new JTextField(new JTextField3_FixedLengthDocument(5),"",5);
        // 针对以上创建的组件以网格组布局方式进行布局
        gbc.gridy=1;
        gbc.gridx=0;
        p1.add(l1,gbc);
        gbc.gridx=1;
        p1.add(t1,gbc);
        gbc.gridy=2;
        gbc.gridx=0;
        p1.add(l2,gbc);
        gbc.gridx=1;
        p1.add(t2,gbc);
        gbc.gridy=3;
        gbc.gridx=0;
        p1.add(l3,gbc);
        gbc.gridx=1;
        p1.add(t3,gbc);
        gbc.gridy=4;
        gbc.gridx=0;
```



```

        p1.add(l4, gbc);
        gbc.gridx=1;
        p1.add(t4, gbc);
        contentPane.add(p1);
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
}

class JTextField3_FixedLengthDocument extends PlainDocument
{
    // 创建 JTextField3_FixedLengthDocument 类, 让它集成 PlainDocument 类
    private int maxLength;
    public JTextField3_FixedLengthDocument(int maxLength)
    // 在构造器方法中设置文本的最大长度
    {
        this.maxLength = maxLength;
    }
    // 此方法是向文档中插入数据
    public void insertString(int offset, String str, AttributeSet att) throws BadLocationException
    {
        // offset 是插入数据的偏移量, str 是插入的数据内容, att 是插入的数据属性
        if ( getLength() + str.length() > maxLength )
        {
            Toolkit.getDefaultToolkit().beep();
        }
        else
        {
            super.insertString(offset, str, att);
        }
    }
}

```

上面程序代码的运行结果如图 11.4 所示。

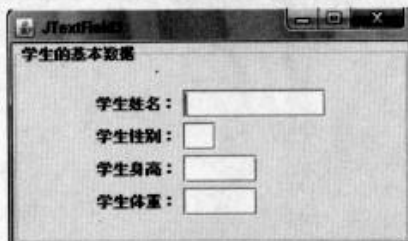


图 11.4 使用 Document 来创建文本域

上述实例的关键是创建一个 `JTextField3_FixedLengthDocument` 类，它是集成 `PlainDocument` 类，只要创建此类，就可以很轻松地创建文本域组件。

组件基本上都有自己的事件处理方式，那么 `JTextField` 文本域是如何处理事件的呢？

可以通过 `JTextField` 类中的 `addActionListener()` 方法检测到用户是否在 `JTextField` 上单击 `Enter` 键，就如同前面所介绍 `JButton` 按下按钮时所产生的事件（`Event`）一样。在 AWT 中的 `TextField` 文本域是有文本变化事件的，而 Swing 中的 `JTextField` 文本域只有一种事件就是动作事件 `ActionEvent`。下面将给出一个实例，该实例是针对上面的例子给出一个文本域的键盘回

车动作事件，其具体代码如下所示：

```
// 这段程序代码主要是处理文本框的键盘事件处理
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
public class test3
{
    ...

    t1.addActionListener(new ActionListener()           // 处理键盘回车动作事件
    {
        public void actionPerformed(ActionEvent Event)
        {
            t1.setText("");                               // 当回车后，t1 文本框会被清空
        }
    });
}
// 创建 JTextField3_FixedLengthDocument 类，让它集成 PlainDocument 类
class JTextField3_FixedLengthDocument extends PlainDocument
{
    ...
}
```

运行上述程序代码后，在文本框中输入数据后回车，文本框中的内容将会被清空。

11.2.2 如何使用 JPasswordField

本小节将讲述密码文本域的用法。在网络中填写登录密码时，密码都会用“*”号代替用户输入的字段，这样可避免用户输入的密码信息被旁人偷窥。而 Swing 中的 JPasswordField 就可以提供这样的功能。JPasswordField 继承自 JTextField 类，因此也可以使用 JTextField 类中的方法，如 addActionListener()、removeActionListener()、setHorizontalAlignment()等。如同 JTextField 一样，JPasswordField 也是一个单行的输入组件，不同的是 JPasswordField 多了屏蔽 (Mask) 的功能，也就是说在 JPasswordField 中的字符都会以单一的字符类型表现出来。在学习如何使用 JPasswordField 之前，先以表格的形式将它的构造器列举出来，如表 11.2 所示。

表 11.2 JPasswordField 构造器

JPasswordField 构造器	说明
JPasswordField()	创建一个空白的密码文本域
JPasswordField(Document doc,String text, int columns)	使用指定的文件存储模式建立一个新的 JPasswordField，并设置其初始化字符串和字段长度
JPasswordField(int columns)	建立一个新的 JPasswordField，并设置其初始字段长度
JPasswordField(String text)	建立一个新的 JPasswordField，并设置其初始化字符串
JPasswordField(String text,int columns)	建立一个新的 JPasswordField，并设置其初始字符串和字段长度

JPasswordField 的构造函数和 JTextField 的构造函数相似，惟一不同的是在 JPasswordField 输入时字符会以屏蔽字符的类型表示。下面将给出一个实例，这个实例主要是设计一个登录窗口，同前面的实例一样，只不过密码框是使用 JPasswordField 组件实现的，其代码如下所示：


```

/*这段程序代码主要是创建一个登录窗口，惟一与本章第一个实例不同的是将原来输入密码的文本域
* 变成了密码文本框组件
*/
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
// 这是一个登录类，设计成一个继承容器的类
class login extends JPanel
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=300;
    static final int HEIGHT=150;
    JFrame loginframe;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    login()
    {
        ...
        // 与上述实例中相同位置的代码相同，这里不再列出
        final JPasswordField passwordinput=new JPasswordField(8);    // 创建一个 JPasswordField 对象
        ...
        // 与上述实例中相同位置的代码相同，这里不再列出
    }
}
class windowsclose extends JPanel
{
    JFrame jf;
    windowsclose()
    {
        jf=new JFrame("确认窗口");
        jf.setVisible(true);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setContentPane(this);
        JLabel l=new JLabel("真的要放弃登录吗? ");
        JLabel l2=new JLabel();
        JButton b1=new JButton("是");
        JButton b2=new JButton("否");
        GridBagConstraints constraints1=new GridBagConstraints();
        constraints1.fill=GridBagConstraints.NONE;
        constraints1.anchor=GridBagConstraints.EAST;
        constraints1.weightx=3;
        constraints1.weighty=3;
        GridBagLayout layout=new GridBagLayout();
        setLayout(layout);
        add(l,constraints1,1,0,1,1);
        add(b1,constraints1,0,1,1,1);    // 使用网格组布局添加组件
        add(b2,constraints1,2,1,1,1);
        add(l2,constraints1,1,1,1,1);
        jf.pack();
        b1.addActionListener(new ActionListener()

```



```

    {
        public void actionPerformed(ActionEvent Event)
        {
            jf.dispose();
        }
    });
    b2.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent Event)
        {
            jf.dispose();
        }
    });
}
public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
{
    constraints.gridx=x;
    constraints.gridy=y;
    constraints.gridwidth=w;
    constraints.gridheight=h;
    add(c,constraints);
}
}
public class test
{
    public static void main(String[] args)
    {
        login log=new login();
    }
}

```

上面程序代码的运行结果如图 11.5 所示。

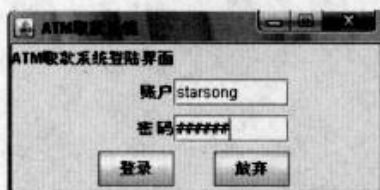


图 11.5 使用密码文本组件 (a)

当输入正确的密码和账户后，单击“登录”按钮后对话框消失，如果输入不正确的信息，对话框如图 11.6 所示。

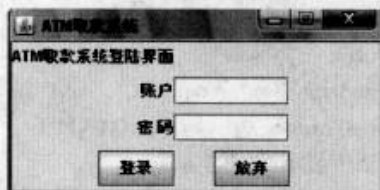


图 11.6 使用密码文本组件 (b)

如果单击“是”按钮，那么两个对话框全部消失，如果单击“否”按钮，那么确认对话框会消失，主对话框依然存在。确认对话框很容易实现。至于密码文本组件的事件处理和文本域是一样的，这里不再赘述。

JFormattedTextField 组件与 JTextField 组件看起来非常相似，但是它们的应用则不同。平时生活中，JFormattedTextField 可以为电话号码提供一个类似于“(###)###-#####”的电话号码格式，例如(086)021-58392851。它不会接受任何不遵循此格式的输入。在较为复杂的情况下，既有显示格式化器，也有输入格式化器。

既然是事先就规定好格式，那么如何配置这些事先规定好的格式呢？日期和数字可以用来自 `java.text` 软件包的 `DateFormat` 和 `NumberFormat` 类指定输入格式。而有的时候可以使用字符来代替，例如上面所讲的“`(###)###-#####`”，这是通过 `MaskFormatter` 使用一系列字符指定可接受的输入来工作。

下面将给出程序代码实例，实例中使用上面介绍过的方式来创建 `JFormattedTextField` 组件，从而让读者能够真正熟悉 `JFormattedTextField` 的用法。其具体的程序代码如下所示：

• 154 •


```
mfl.setPlaceholderCharacter('_');
JFormattedTextField ftf2 = new JFormattedTextField(mf1);
content.add(ftf2);
MaskFormatter mf2 = new MaskFormatter("###-###-####");
// 通过掩码的形式设置文本框中显示数据的格式
JFormattedTextField ftf3 = new JFormattedTextField(mf2);
// 创建一个事先设置好格式的文本框
content.add(ftf3);
f.setSize(300, 100);
f.setVisible(true);
}
```

上面程序代码的运行结果如图 11.7 所示。

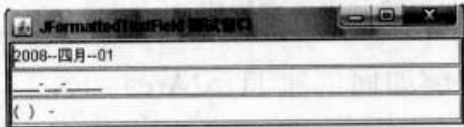


图 11.7 JFormattedTextField 的用法

上述程序规定了三种不同的格式，如果输入的信息不遵从规定的格式，那么 JFormattedTextField 会将不规范的输入转换成规范的输入。

在实际开发中，JFormattedTextField 类一般用来对规范格式的约束。普通文本域的三种不同种类，彼此之间有其共同性，也有其不同点。JTextField 一般用的最广泛，作为用户单行输入的文本框，JPasswordField 用于密码框。JFormattedTextField 用于规范格式的单行文本的输入。

11.3 如何使用文本区组件

当用户的输入信息文本超过一行，可以选择使用文本区组件（JTextArea）来容纳它。在一个文本区的组件内用户可以输入多行文本，换行的方式就是采用默认的回车，当回车时每行的文本都会以“\n”字符作为结束标志。下面将以表格的方式列出 JTextArea 组件的构造器，如表 11.3 所示。

表 11.3 JTextArea 组件的构造器

JTextArea 组件的构造器	说明
JTextArea()	建立一个新的 JTextArea
JTextArea(Document doc)	使用指定的文件存储模式建立一个新的 JTextArea
JTextArea(Document doc,String text, int row ,int columns)	使用指定的文件存储模式建立一个新的 JTextArea 并设置其初始字符串、列、字段长度
JTextArea(int row,int columns)	建立一个新的 JTextArea 并设置其初始列、字段长度
JTextArea(String text)	建立一个新的 JTextArea 并设置其初始字符串
JTextArea(String text,int row,int columns)	建立一个新的 JTextArea 并设置其初始字符串、列、字段长度

JTextArea 组件构造器与 JTextField 组件的构造器差不多，惟一不同的就是一个是多行输入，一个是单行输入，下面就将介绍一些在实际开发中经常遇到的问题以及解决方法，如表 11.4 所示。

表 11.4 JTextArea 组件遇到的问题和解决方法

问题	解决方法
setLineWrap(Boolean b)	当文本区中的内容超过了显示范围，多余的内容会被文本区剪切掉，可以通过此方法来决定超过的内容是否被剪切。如果不被剪切的话，那么会对超过内容进行自动换行
文本区并没有提供滚动条	如果想让文本区能够提供滚动条，则必须把该文本区放到一个滚动面板中，再把滚动面板放到内容面板中

接下来将给出一个实例来讲解如何创建 JTextArea 组件。其程序代码如下：

```
// 这段程序主要是创建一个文本区组件，然后使用这个组件的复制、粘贴、剪切三个功能
import java.awt.event.*;
import javax.swing.*;

public class test6 implements ActionListener
{
    JTextArea textarea=null;
    JButton button1,button2,button3;
    public test6()
    {
        JFrame f=new JFrame("文本区测试窗口");
        Container contentPane=f.getContentPane();
        contentPane.setLayout(new BorderLayout());
        textarea=new JTextArea(10,15);
        // 创建一个文本区对象，此对象长度为 10 个字符，宽度为 15 个字符
        JScrollPane scrollPane=new JScrollPane(textarea);
        // 将文本区放入到滚动条组件中，使得文本区组件可以滚动观察
        JPanel panel=new JPanel();
        panel.setLayout(new GridLayout(1,3));
        button1=new JButton("复制");
        button1.addActionListener(this);
        button2=new JButton("粘贴");
        button2.addActionListener(this);
        button3=new JButton("剪切");
        button3.addActionListener(this);
        panel.add(button1);
        panel.add(button2);
        panel.add(button3);
        contentPane.add(scrollPane,BorderLayout.CENTER);
        contentPane.add(panel,BorderLayout.SOUTH);
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        })
    }
}
```



```

    });
}
public static void main(String[] args)
{
    new test6();
}
public void actionPerformed(ActionEvent e)    // 此方法为按钮组件的动作时间
{
    if (e.getSource()==button1)                // 当单击"复制"按钮, 则执行复制功能
    {
        textarea.copy();
    }
    if (e.getSource()==button2)                // 当单击"粘贴"按钮, 则执行粘贴功能
    {
        textarea.paste();
    }
    if (e.getSource()==button3)
    {
        textarea.cut();                        // 当单击"剪切"按钮, 则执行剪切功能
    }
}
}

```

上面程序的运行结果如图 11.8 所示。



图 11.8 文本区实例 (a)

当输入一段文本“我是程序员”后, 选中它并单击“复制”按钮, 而后单击“粘贴”按钮, 结果如图 11.9 所示。

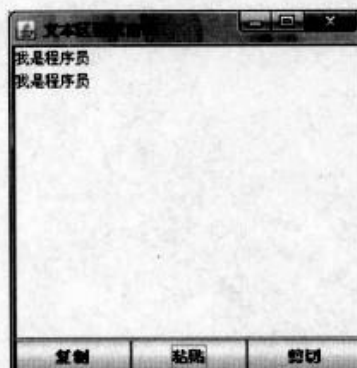


图 11.9 文本区实例 (b)

当选中“我是程序员”后, 单击“剪切”按钮, 而后单击“粘贴”按钮, 结果如图 11.10 所示。



图 11.10 文本区实例 (c)

以上实例利用了文本组件的复制、粘贴和剪切三个方法，实现了复制、粘贴和剪切三种不同的功能。下面将为读者讲述文本区的事件处理。

在 `JTextArea` 中使用的 `Listener` 有两种，一个是 `UndoableEditListener`，另一个是 `DocumentListener`。

- `UndoableEditListener` 接口负责记录 `JTextArea` 中所有操作发生的顺序，并且可以运行还原上一步的功能。这个功能在目前的软件中应用相当广泛，如文书编辑软件 Word 中的复原功能、小画家软件中的复原功能。
- `DocumentListener` 接口则是记录发生在 `JTextArea` 中所有的事件（如键入字符、删除字符、剪下、贴上）并将所有的事件以树状的层次式结构组织起来，也就是说当 `JTextArea` 中的内容有任何变动时，会激活 `DocumentEvent` 事件，此时必须使用 `DocumentListener` 接口中的方法来处理此事件。

下面将通过程序代码为读者讲述文本区的事件监听的处理过程。其程序代码如下所示：

```
// 这段程序代码主要创建文本区组件，在文本区中处理了有关事件处理
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.undo.*;
import javax.swing.event.*;
import javax.swing.text.*;

public class test7 extends JFrame implements UndoableEditListener, DocumentListener
{
    private UndoableEdit edit;
    // 表示编辑的对象，该编辑已完成并且可以对其进行撤消和恢复操作
    private JTextArea jta;
    private JTextArea message;
    private JMenu undoitem;
    private JMenu redoitem;
    public test7()
    {
        super("文本区事件处理测试窗口");
        jta = new JTextArea();
        jta.getDocument().addUndoableEditListener(this);
        jta.getDocument().addDocumentListener(this);
        message = new JTextArea();
        message.setEditable(false);
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(1,1));
```



```

p1.setBorder(BorderFactory.createTitledBorder("Edit Area"));
/* BorderFactory.createTitledBorder 创建一个新标题边框，使用默认边框（浮雕化）、
 * 默认文本位置（位于顶线上）、默认调整（Leading）以及由当前外观确定的默认字体和文本颜色
 * 并指定了标题文本
 */
p1.add(new JScrollPane(jta));
JPanel p2 = new JPanel();
p2.setLayout(new GridLayout(1,1));
p2.setBorder(BorderFactory.createTitledBorder("Message"));
p2.add(new JScrollPane(message));
getContentPane().setLayout(new GridLayout(2,1));
getContentPane().add(p1);
getContentPane().add(p2);
JMenuBar bar = new JMenuBar();
JMenu theMenu = new JMenu("Edit");
undoitem = new JMenuItem("Undo");
redoitem = new JMenuItem("Redo");
theMenu.add(undoitem);
theMenu.add(redoitem);
bar.add(theMenu);
// 创建菜单项
updateMenuItem();
setJMenuBar(bar);
setSize(300,300);
undoitem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ev)
    {
        edit.undo();                // 撤销所有操作
        updateMenuItem();
        message.append("- Undo -\n");
    }
});
redoitem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ev)
    {
        edit.redo();                // 假定编辑已被撤销，则重新应用该编辑
        updateMenuItem();
        message.append("- Redo -\n");
    }
});
} // end of JTextArea5
public void undoableEditHappened(UndoableEditEvent ev)
{
    StringBuffer buf = new StringBuffer(200);
    edit = ev.getEdit();
    buf.append("undoableEdit:");
    buf.append(edit.getPresentationName());
    buf.append("\n");
    message.append(buf.toString());
    updateMenuItem();
} // 取消操作的事件处理
public void updateMenuItem()
{

```



```

        if (edit != null)
        {
            undoitem.setEnabled(edit.canUndo());
            redoitem.setEnabled(edit.canRedo());
            undoitem.setText(edit.getUndoPresentationName());
            redoitem.setText(edit.getRedoPresentationName());
        }
        Else
        {
            undoitem.setEnabled(false);
            redoitem.setEnabled(false);
            undoitem.setText("Undo");
            redoitem.setText("Redo");
        }
    }
    // 菜单变化后的结果
    public void showDE(DocumentEvent de)
    {
        StringBuffer debuf=new StringBuffer(100);
        debuf.append(de.getType());
        debuf.append("Offset:");
        debuf.append(de.getOffset());
        debuf.append("Length:");
        debuf.append(de.getLength());
        Element Eroot=jta.getDocument().getDefaultRootElement();
        DocumentEvent.ElementChange Echange=de.getChange(Eroot);
        if (Echange==null)
        {
            debuf.append("(No Element Change)");
            // 获得给定元素的更改信息
        }
        Else
        {
            debuf.append("Element Change:index");
            debuf.append("Echange.getIndex()");
        }
        debuf.append("\n");
        message.append(debuf.toString());
    }
    public void changedUpdate(DocumentEvent de)
    {
        // 显示出信息
        showDE(de);
    }
    public void insertUpdate(DocumentEvent de)
    {
        showDE(de);
    }
    public void removeUpdate(DocumentEvent de)
    {
        showDE(de);
    }
    public static void main(String[] args)
    {
        JFrame f=new test7() ;
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)

```



```
    {  
        System.exit(0);  
    }  
});  
f.setVisible(true);  
}  
}
```

上面程序代码的运行结果如图 11.11 所示。



图 11.11 文本区事件处理 (a)

当在 Edit Area 选项组中输入信息时，其结果如图 11.12 所示。

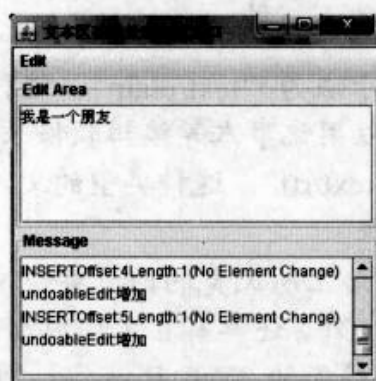


图 11.12 文本区事件处理 (b)

当进行取消操作时，其结果如图 11.13 所示。

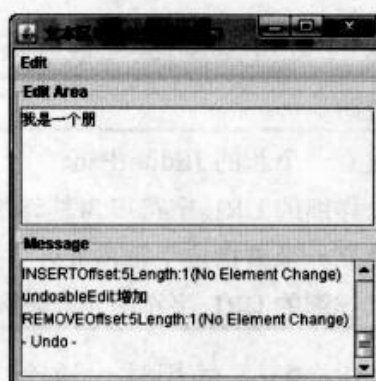


图 11.13 文本区事件处理 (c)

当进行重做操作时，其结果如图 11.14 所示。



图 11.14 文本区事件处理 (d)

上面已经详细讲述了如何创建文本区以及如何处理文本区的事件。在实例程序中涉及到了很多有关 Document 方面的知识，有兴趣的读者可以参考一些此方面的书籍。

下面主要为读者介绍两种带格式的文本区，一种是 JEditorPane，另一种是 JTextPane。JEditorPane 的最主要功能在于展现不同类型的文件格式内容。JTextPane 提供了许多对文字的处理，如改变颜色、字体缩放、文字风格、加入图片等。下面将针对这两种文本区给予详细讲解。

1. 如何使用 JEditorPane

JEditorPane 支持的文件类型有如下三种。

- 纯文本类型：其类型的表示法为“text/plain”，这种类型的文件就是最常使用的文本文件，这种类型的文件可以用记事本等编辑软件来编辑。
- RTF 类型：其表示法为“text/rtf”，这种类型的文件特色是能对文字内容进行字体缩放、变形、上色等特殊效果。
- HTML 类型：也就是在网络上所浏览的网页类型，其表示法为“text/html”，这类文件除了对字体效果的表现之外，还具有在文件内加入图片、超链接等相关功能。但是 JEditorPane 并不是一个全功能的 Web Browser，它仅能支持简单的 HTML 语法。JEditorPane 支持 HTML 类型的文件最主要的用途是制作在线辅助说明文件。

JEditorPane 的构造器说明如表 11.5 所示。

表 11.5 JEditorPane 构造器

JEditorPane 构造器	说明
JEditorPane()	建立一个新的 JEditorPane
JEditorPane(String url)	以详细的 URL 字符串为基础建立一个 JEditorPane
JEditorPane(String type,String text)	建立一个被指定字符串 text 并指定初始化 JEditorPane 的类型
JEditorPane(URL initialPage)	以详细的 URL 字符串当作输入值来建立一个 JEditorPane

下面将给出实例使读者熟悉 JEditorPane 的用法，这个实例主要是通过 JEditorPane(String type,String text)构造器来创建一个 JEditorPane 组件，其具体的实例代码如下：

```
// 这段程序代码主要是为读者展示如何创建一个文本格式的 JEditorPane 组件
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
```



```

public class test8
{
    public static void main(String[] args)
    {
        String str=new String("我是一名程序员.\n 我是一名优秀的程序员!\n 我是一名优秀的 JAVA 程序员!");
        JEditorPane editPane=new JEditorPane("text/plain",str);
        // 创建一个 JEditorPane 组件, 以 str 为内容, 以 text/plain 为格式
        editPane.setEditable(false);
        JFrame f=new JFrame("JEditorPane2");
        f.setContentPane(new JScrollPane(editPane));
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
}

```

上面程序代码的运行结果如图 11.15 所示。

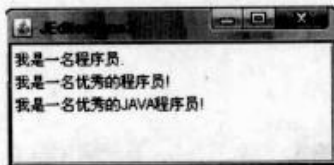


图 11.15 创建一个 JEditorPane 组件 (a)

上面是一个简单的创建文本类型的 JEditorPane 组件, 下面再列举一个实例, 该实例主要是使用 JEditorPane(URL initialPage)构造器来创建一个 JEditorPane 组件, 其具体实例代码如下所示:

```

// 这段程序代码主要向读者展示如何创建一个带 URL 地址的 JEditorPane 组件
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class test9
{
    public static void main(String[] args)
    {
        JEditorPane editPane=null;
        Try
        {
            URL address=new URL("http:// www.google.com");
            editPane=new JEditorPane(address);          // 创建一个带 URL 地址的 JEditorPane 组件
        }
        catch(MalformedURLException e)
        {
            System.out.println("Malformed URL:"+e);
        }
        catch(IOException e)

```



```
{
    System.out.println("IOException:"+e);
}
editPane.setEditable(false);
JFrame f=new JFrame("网页测试窗口");
f.setContentPane(new JScrollPane(editPane));
f.setSize(200,250);
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
```

上面程序的运行结果如图 11.16 所示。

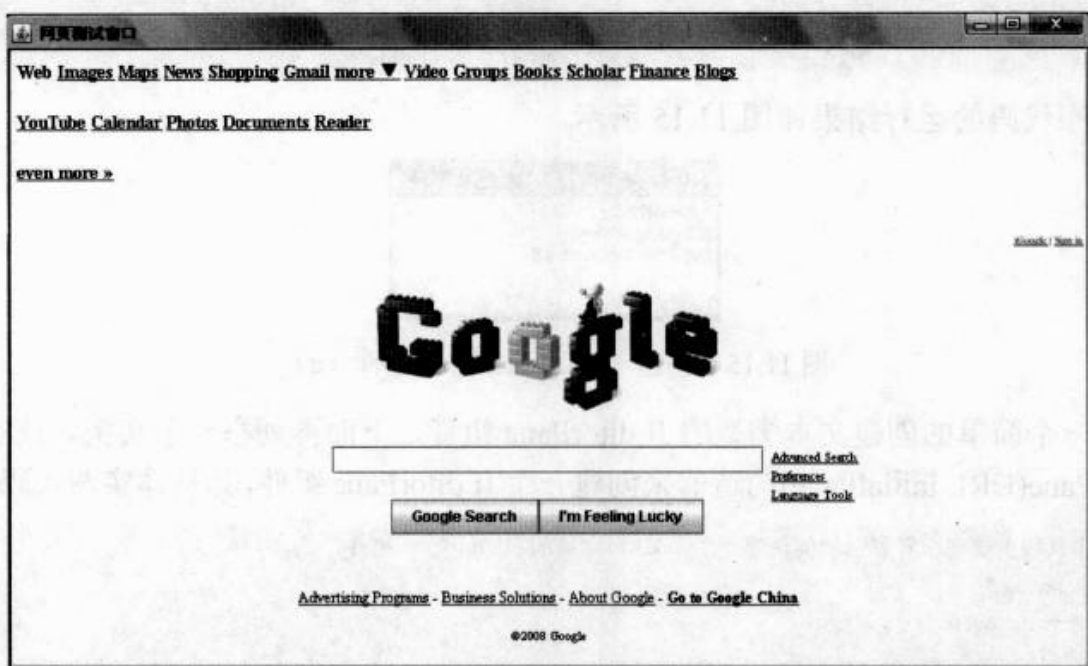


图 11.16 创建一个 JEditorPane 组件 (b)

上面的程序创建了一个具有网页页面的 JEditorPane 组件，但是在实际开发中，不可能只创建一个网页就结束了，一定会牵涉到链接等基本操作，这就要使用到组件的事件处理。限于篇幅，这里不再进行介绍，有兴趣的读者可以自行查阅相关的资料。

2. 如何使用 JTextPane

JTextPane 继承自 JEditorPane，是专为文字和版面处理设计的组件，类似于办公软件包中的 Word 软件。JTextPane 提供了许多对文字的处理，如改变颜色、字体缩放、文字风格、加入图片等。它有一些常用的方法，如 `getDocument()`、`getParagraphAttributes()`、`setDocument(Document doc)`、`setStyledDocument(StyledDocument doc)`、`setParagraphAttributes(AttributeSet attr, boolean replace)`，这些方法在实际开发中，使用的频率较高。希望读者能够熟悉它们的用法。JTextPane 组件的构造器的说明如表 11.6 所示。

表 11.6 JTextPane 构造器

JTextPane 构造器	说明
JTextPane()	建立一个新的 JTextPane
JTextPane(StyledDocument doc)	以指定的文件模式建立一个新的 JTextPane

JTextPane 对可输入区域内容的设计概念是一个类似于 Word 的设计概念，也就是说在 JTextPane 中的文字结构是有段落概念的。段落的概念就是以回车键为每一段落的分界点，每按一次回车键就增加一个段落。在 JTextPane 中则是以整个编辑区为根节点、每个段落为枝节点、每个字符为叶节点来存储文件，因为 JTextPane 是采用这样的方式来存储数据，因此，在 JTextPane 中也可以像 Word 文件一样将各个段落设置成不同的属性，如第一段为斜体字、字体大小为 14 号字、粗体字，第二段为斜体字、字体颜色为蓝色、向左边界缩排 2 厘米等。另外，还可以设置 JTextPane 编辑区内输入的文字与各个边界间的距离。由这些功能可知，对于一个 TextComponent 来说 JTextPane 是一个具有相当多实用功能的组件。

下面将列举一个实例，通过实例来熟悉 JTextPane 组件的使用方法，实例代码如下：

```
// 这段程序主要是向读者展示如何创建 JTextPane 组件
// 并且像 Word 文档一样，可以修改文字属性
import javax.swing.*;
import javax.swing.text.*;
import java.awt.event.*;
import java.awt.*;
public class test10
{
    private JTextPane textPane;
    public test10()
    {
        textPane=new JTextPane();
        textPane.setBackground(Color.white);
        textPane.setEditable(false);
    }
    public void setRED_Bold_20(String str)
    {
        SimpleAttributeSet attrset=new SimpleAttributeSet();
        StyleConstants.setForeground(attrset,Color.red);
        StyleConstants.setBold(attrset,true);
        insert(str,attrset);
    }
    public void setGREEN_Italic_Bold_22(String str)
    {
        SimpleAttributeSet attrset=new SimpleAttributeSet();
        StyleConstants.setForeground(attrset,Color.green);
        StyleConstants.setItalic(attrset,true);
        StyleConstants.setFontSize(attrset,21);
        insert(str,attrset);
    }
    // 此方法主要设置 str 字符串的属性，例如属性值、颜色，然后将 str 插入到 JTextPane 中
    public void setBlack_UnderLine_Italic_24(String str)
    {
        SimpleAttributeSet attrset=new SimpleAttributeSet();
        StyleConstants.setForeground(attrset,Color.black);
        StyleConstants.setUnderline(attrset,true);
        StyleConstants.setItalic(attrset,true);
    }
}
```

// 创建一个属性集
// 设置前景色为红色
// 设置字体为粗体

// 创建一个属性集
// 设置前景色为绿色
// 设置字体
// 设置字体大小

// 创建一个属性集
// 设置前景色为黑色
// 设置文字带下划线
// 设置字体


```

        StyleConstants.setFontSize(attrset,32);
        insert(str,attrset);
    }
    public void insert(String str,AttributeSet attrset)
    // 这个方法主要是将字符串插入到 JTextPane 中
    {
        Document docs=textPane.getDocument();
        // 利用 getDocument()方法取得 JTextPane 的 Document instance.0
        str=str+"\n";
        try
        { docs.insertString(docs.getLength(),str,attrset);
        }
        catch(BadLocationException ble)
        {
            System.out.println("BadLocationException:"+ble);
        }
    }
    public Component getComponent()
    {
        return textPane;
    }
    public static void main(String[] args)
    {
        test10 pane=new test10();
        pane.setRED_Bold_20("这是一本有关 JAVA 图形编程的书籍");
        pane.setGREEN_Italic_Bold_22("这是一本软件编程的书籍");
        pane.setBLACK_UnderLine_Italic_24("这是一本有关 JAVA 软件编程的书籍");
        JFrame f=new JFrame("JTextPane1");
        f.getContentPane().add(pane.getComponent());
        f.setSize(450,180);
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
}

```

上面程序代码的运行结果如图 11.17 所示。

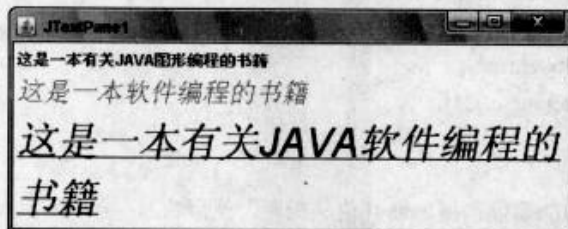


图 11.17 JTextPane 组件的创建

限于篇幅，这里仅对 JTextPane 组件进行了简单介绍，有兴趣的读者可以查阅相关资料。

11.4 如何打印文本组件

其实打印文本组件非常简单，下面为读者介绍几种打印方法，如表 11.7 所示。

表 11.7 打印方法一览表

打印方法	说明
<code>public boolean print()</code>	显示打印对话框，然后在交互模式下打印此 <code>JTextComponent</code> ，不打印标题和脚注文本
<code>public boolean print(MessageFormat headerFormat, MessageFormat footerFormat)</code>	显示打印对话框，然后在交互模式下打印此 <code>JTextComponent</code> ，打印指定的标题和脚注文本。 <code>headerFormat</code> 在 <code>MessageFormat</code> 中用作标题的文本，或者为 <code>null</code> ，表示没有标题； <code>footerFormat</code> 在 <code>MessageFormat</code> 中用作脚注的文本，或者为 <code>null</code> ，表示没有脚注
<code>print(MessageFormat headerFormat, MessageFormat footerFormat, boolean showPrintDialog, PrintService service, PrintRequestAttributeSet attributes, boolean interactive)</code>	打印此 <code>JTextComponent</code> 的内容， <code>headerFormat</code> 在 <code>MessageFormat</code> 中用作标题的文本，或者为 <code>null</code> ，表示没有标题； <code>footerFormat</code> 在 <code>MessageFormat</code> 中用作脚注的文本，或者为 <code>null</code> ，表示没有脚注； <code>showPrintDialog</code> 如果显示打印对话框，则为 <code>true</code> ，否则为 <code>false</code> ； <code>service</code> 初始 <code>PrintService</code> ，或者为 <code>null</code> ，表示使用默认值； <code>attributes</code> 应用于打印作业的作业属性，或者为 <code>null</code> ，表示没有作业属性； <code>interactive</code> 表示是否在交互模式下进行打印

针对上面的方法，下面将给出一个打印实例，这个实例用于创建一个文本区对象，然后将其中的内容打印出来，其具体的程序代码如下：

```
// 这段程序代码主要是向读者展示如何打印文本组件
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
public class test11 extends JPanel
{
    static JTextArea text;
    test11()
    {
        JFrame frame=new JFrame("打印测试程序");
        frame.setContentPane(this);
        frame.setVisible(true);
        text=new JTextArea(10,10);
        JButton button=new JButton("打印文档");
        setLayout(new FlowLayout());
        add(text);
        add(button);
        button.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent Event)
            {
                try
```



```
        {  
            text.print();// 调用文本的打印方法  
        }  
        catch(Exception e){}  
    }  
    });  
}  
public static void main(String args[])  
{  
    new test11();  
}
```

运行上述程序会调用系统的打印系统，从而针对文本进行打印。但上述程序只是调用了最简单的 `print()` 方法，读者可以尝试使用其他两个打印方法来打印文本组件，并观察最后的结果。

11.5 本章小结

本章主要讲述了文本组件的使用，如 `JTextField`、`JTextArea`、`JTextPane`、`JEditorPane`、`JPasswordField` 等组件。针对每个不同的组件的事件进行处理，并配合实例进行详细讲述。力求让读者能够真正掌握其用法。在实际开发中，文本组件的使用频率非常高，几乎每个软件工程中都会涉及到，所以希望读者通过练习能够真正掌握它们。

11.6 本章习题

1. 设计一个程序，使用 `JTextField` 组件、标签组件创建一个简单的输入系统，当在 `JTextField` 组件中输入数据时，使用回车来代替按钮组件的动作事件。

要求：

(1) 其最终效果如图 11.18 所示。



图 11.18 本章习题 1 (a)

(2) 当在文本组件中输入数据并按回车键后，将出现如图 11.19 所示的结果。

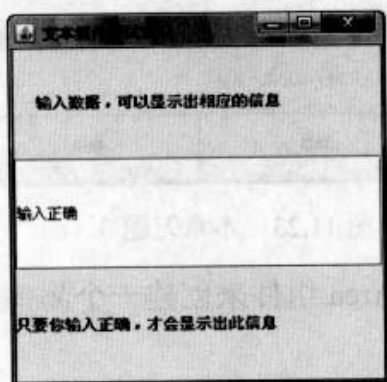


图 11.19 本章习题 1 (b)

2. 设计一个程序，在程序中创建一个菜单，通过菜单中的“打印”命令，将面板中的 JTextArea 组件中的内容打印出来。

要求：其最终效果如图 11.20 所示。



图 11.20 本章习题 2

3. 设计一个程序，在程序中设计一个密码系统，第一级密码框使用 JPasswordField 组件，第二级密码框使用 JFormattedTextField 组件，必须按照一定的格式输入密码才能显示出成功登录的界面。

要求：

(1) 其最终效果如图 11.21 所示。



图 11.21 本章习题 3 (a)

(2) 如果密码正确并按回车键确认后，其效果如图 11.22 所示。



图 11.22 本章习题 3 (b)

(3) 如果密码依然正确, 按回车键确认后, 其效果如图 11.23 所示。



图 11.23 本章习题 3 (c)

4. 设计一个程序, 使用 JTextArea 组件来创建一个简单的文本编辑器, 具有复制、粘贴、剪切、打印等功能。

要求: 其最终效果如图 11.24 所示。

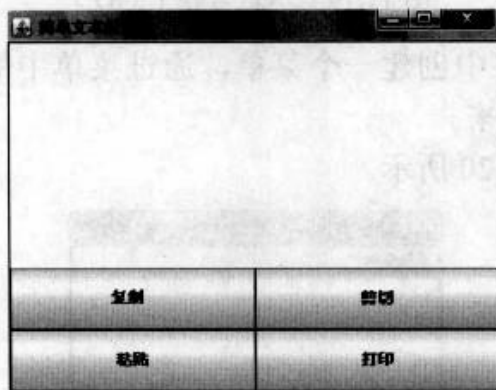


图 11.24 本章习题 4

第12章 如何使用窗口、对话框和JApplet组件

本章主要为读者讲述顶层容器类的用法。所谓顶层容器类就是指那些能够容纳其他容器或者组件的容器类，它们是整个软件的最顶层，可以独立显示，而无须依赖其他容器类。顶层容器类包括窗口、对话框、JApplet。窗口一般是指 JFrame 类。而所谓的对话框，就像平时打开文件、保存文件以及弹出的确认对话框等。至于 JApplet，则是指那种经常使用在网页中的小程序，也就是 java.applet.Applet 的扩展版。在这一章中，将会为读者详细讲述这些类的使用。

12.1 如何使用窗口组件

本节将讲述顶层容器类中窗口的知识，窗口类的结构层次如图 12.1 所示。

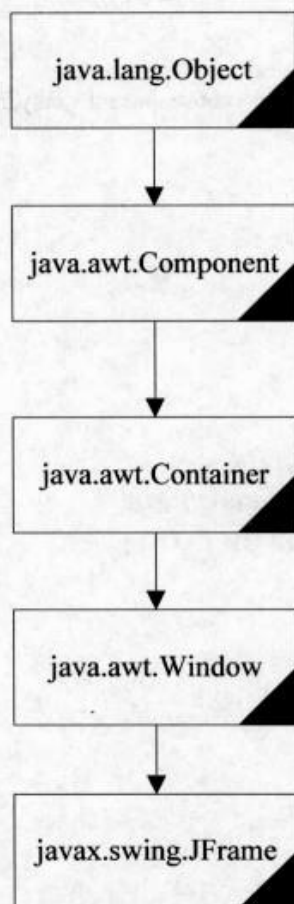


图 12.1 窗口类的结构层次图

从上面的结构层次图可以看出，它属于容器类，并且继承了 Window 类对象可以独立存在的特性，从而注定了 JFrame 框架容器可以独立存在。窗口的应用很简单，要想使用窗口组件，首先必须要通过窗口组件的构造器来创建它。窗口类的构造器说明如表 12.1 所示。

表 12.1 JFrame 构造器

JFrame 构造器	说明
JFrame()	创建一个空白的顶层窗口类
JFrame(String title)	创建一个带标题的空白顶层窗口类
Jframe(GraphicsConfiguration gc)	以屏幕设备指定的 GraphicsConfiguration 和空白标题创建一个 Frame
Jframe(String title, GraphicsConfiguration gc)	创建一个具有指定标题和指定屏幕设备的 GraphicsConfiguration 的 JFrame

下面将通过一个实例来列举使用上面的构造器。这个实例主要是创建一个 JFrame 窗口，在窗口中添加中间容器类，再在中间容器类中添加基本组件。这个实例虽然简单，但是可以通过对这个实例的分析，来使读者更加熟悉窗口类的创建和使用，其实例代码如下：

```
// 这段程序代码主要是为读者展示如何创建一个顶层容器，也就是窗口
import javax.swing.*;
import java.awt.*;
public class JFrametest1 extends JPanel
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=700;
    static final int HEIGHT=400;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    JFrametest1()
    {
        JFrame f=new JFrame("纺织厂职工管理系统");
        // 使用 JFrame(String str)的构造器来创建顶层容器类
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(WIDTH,HEIGHT);
        f.setVisible(true);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        f.setLocation(x,y);
        f.setContentPane(this);
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
        JLabel l=new JLabel("职工信息查询窗口");
        JLabel ll=new JLabel("按照职工名字查询");
```

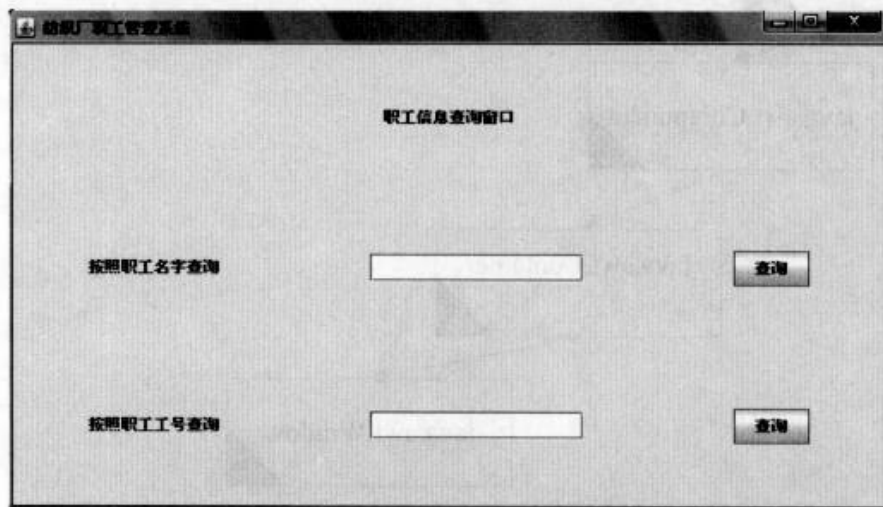


```

JLabel l2=new JLabel("按照职工工号查询");
JTextField tf1=new JTextField(15);
JTextField tf2=new JTextField(15);
JButton b=new JButton("查询");
JButton b1=new JButton("查询");
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=3;
constraints.weighty=3;
add(l,constraints,0,0,3,1);
add(l1,constraints,0,1,1,1);
add(l2,constraints,0,2,1,1);
add(tf1,constraints,1,1,1,1);
add(tf2,constraints,1,2,1,1);
add(b,constraints,2,1,1,1);
add(b1,constraints,2,2,1,1);
}
public static void main(String[] args)
{
    new JFrametest1();
}
}

```

上面程序代码的运行结果如图 12.2 所示。



12.2 窗口的创建

从上面的程序代码中可以总结出创建顶层框架窗口的固定格式：

```

static final int WIDTH=700;
static final int HEIGHT=400;
f.setSize(WIDTH,HEIGHT);

```

上面的代码表示设置窗口的大小，以下代码用于设置窗口的位置位于整个屏幕的中心：

```

Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);

```


读者可以将上面两段代码牢记，每次创建窗口的时候，都可以使用这些代码。
一般来说，针对创建顶层容器 JFrame 类可以有两种常用的创建方式，如表 12.2 所示。

表 12.2 创建 JFrame 类的方法

创建 JFrame 类的方法	说明
<code>JFrame frame=new JFrame();</code> <code>Frame.setTitle(顶层容器类对象的名称);</code>	先创建一个空框架，然后给框架赋值
<code>JFrame frame=new JFrame(顶层容器类对象的名称);</code>	直接创建一个带名称的框架

本节介绍了一些创建窗口 JFrame 类的基本方法和常用的一些技巧，希望读者能够牢牢掌握它，并且不断创新，从而能够设计出比较专业的程序代码。

12.2 如何使用对话框组件

对话框系统是指用来作为提醒性的交互系统，例如在实际操作 Windows 系统中出现错误，会弹出一个错误提示，那就是对话框。对话框的层次结构如图 12.3 所示。

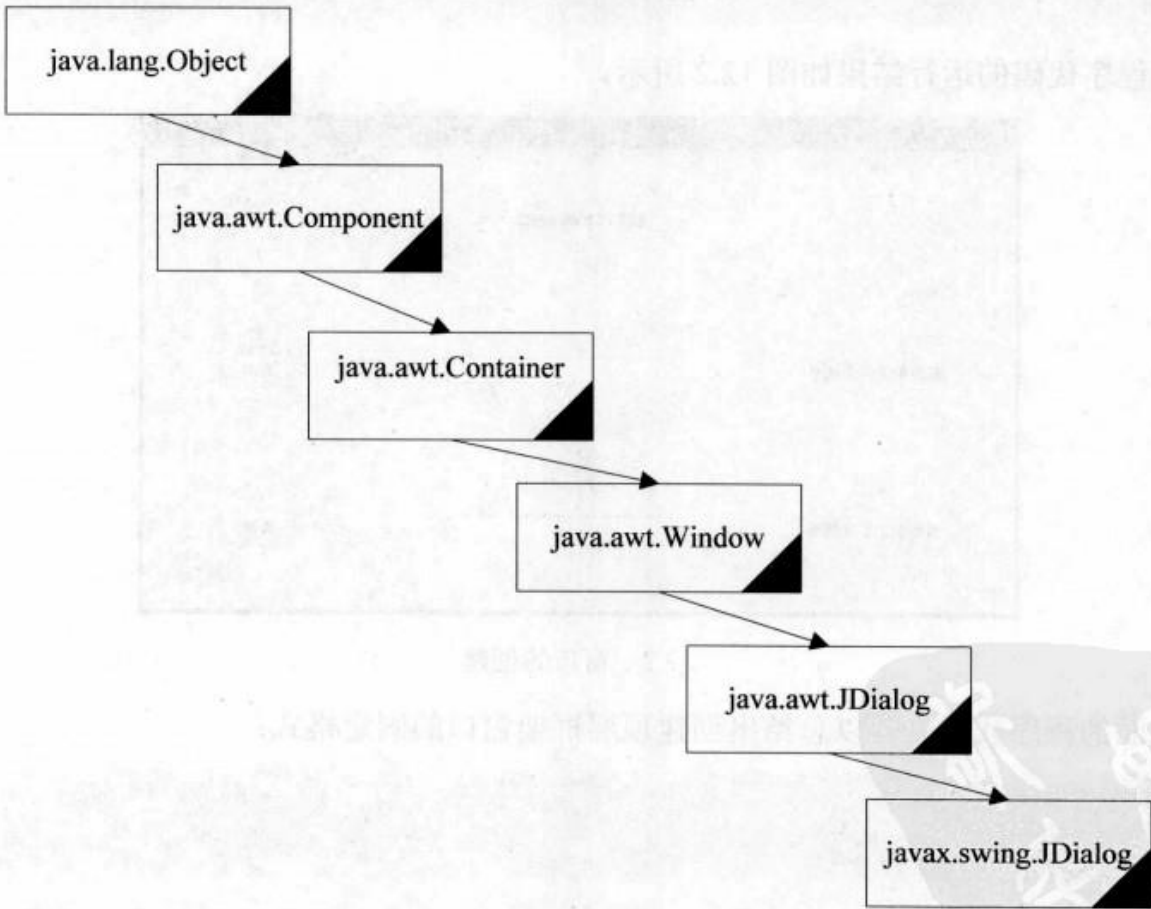


图 12.3 对话框类的层次结构图

由以上结构图可以知道，JDialog 类也是一个类似于 JFrame 类的顶层容器类，Java 提供了 JDialog 与 JOptionPane 两个类来创建对话框系统，事实上，当使用 JOptionPane 时，系统会自动产生 JDialog 组件，并将 JOptionPane 的内容放入 JDialog 的 ContentPane 中，而这些均由系统自动运行，并不需要开发人员介入。使用 JOptionPane 的好处是此组件已经默认了许多交互方式，开发人员只须设置想要的显示模式，JOptionPane 就能轻易地显示出来，这对于开发人

员来说相当方便。若这些模式还是无法满足用户的需求,则可以使用 JDialog 来自行设计对话框。JOptionPane 类的构造器的说明如表 12.3 所示。

表 12.3 JOptionPane 构造器

JOptionPane 构造器	说明
JOptionPane()	建立一个显示测试信息的 JOptionPane 组件
JOptionPane(Object message)	建立一个显示特定信息的 JOptionPane 组件
JOptionPane (Object message,int messageType)	建立一个显示特定信息的 JOptionPane 组件,并设置信息类型
JOptionPane(Object message, int messageType,int optionType)	建立一个显示特定信息的 JOptionPane 组件,并设置信息与选项
JOptionPane (Object message,int messageType, int optionType,Icon icon)	建立一个显示特定信息的 JOptionPane 组件,并设置信息与选项且可显示出图案
JOptionPane(Object message,int messageType, int optionType,Icon icon,Object[] options)	建立一个显示特定信息的 JOptionPane 组件,并设置信息与选项,且可显示出图案。选项值是一个 Object Array,可用作更改按钮上的文字
JOptionPane(Object message,int messageType, int optionType,Icon icon,Object[] options, Object initialValue)	建立一个显示特定信息的 JOptionPane 组件,并设置信息与选项类型,且可以显示出图案。选项值是一个 Object Array,可用作更改按钮上的文字,并设置默认按钮

需要注意的是使用 JOptionPane 对象所得到的对话框必须先关闭对话框窗口,才能返回到产生对话框的母窗口上。

JOptionPane 类有 4 个常用的静态方法,用来显示一些简单的对话框框架,如表 12.4 所示。

表 12.4 JOptionPane 类有 4 个常用的静态方法

JOptionPane 类有 4 个常用的静态方法	说明
showMessageDialog	显示一条信息并且等待用户单击 OK 按钮
showConfirmDialog	显示一条消息并得到确认(同 OK/Cancel 类似)
showOptionDialog	显示一条消息并得到用户在一组选项中的选择
showInputDialog	显示一条消息得到用户的一行输入

当然, JOptionPane 类中远远不止这些静态方法,希望有兴趣的读者自行查阅相关资料。在上面的 4 种静态方法中,每一种静态方法都有几种不同的重载方法,以提供在不同的环境下使用不同风格的对话框。

一个典型的对话框差不多都包括下面几个部分:图标、消息、一个或多个选项按钮、对话框的标题等,这些都是上述静态方法中的参数。

对于图标可以自定义,也可以采用系统默认的图标。如果采用系统默认的图标,系统总共有 5 种不同的消息对应 5 种不同的图标,如表 12.5 所示。



表 12.5 系统默认的图标所对应的消息

系统默认的图标所对应的消息	说明
ERROR_MESSAGE	用于错误消息
INFORMATION_MESSAGE	用于信息消息
WARNING_MESSAGE	用于警告消息
QUESTION_MESSAGE	用于问题
PLAIN_MESSAGE	未使用图标

对话框底部的按钮取决于对话框类型和选项类型。当调用对话框时，可能会出现不同的按钮，所以可以根据不同的需要选择如下 4 种选项类型，如表 12.6 所示。

表 12.6 按钮选项类型

按钮选项类型	说明
DEFAULT_OPTION	外观不应该提供任何选项的类型
YES_NO_OPTION	用于 showConfirmDialog 的类型
YES_NO_CANCEL_OPTION	用于 showConfirmDialog 的类型
OK_CANCEL_OPTION	用于 showConfirmDialog 的类型

当然还有很多不同的选项类型，希望读者能够自行查阅相关资料，多多练习。每一个静态方法的完整形式如表 12.7 所示。

表 12.7 JOptionPane 常用静态方法的完整形式

JOptionPane 常用静态方法的完整形式	说明
Public static void showMessageDialog (Component parentComponent, Object message)	parentComponent 用于确定在其中显示对话框的 框架；如果为 null 或者 parentComponent 不具有框架，则使用默认的框架；message 为要显示的 Object
public static int showConfirmDialog (Component parentComponent, Object message, String title, int optionType)	title 为对话框的标题字符串；optionType 为对话框的选项参数：YES_NO_OPTION、YES_NO_CANCEL_OPTION 或 OK_CANCEL_OPTION
public static int showOptionDialog (Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)	messageType 为消息种类的整数，可以用于确定来自可插入外观的图标：ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE 或 PLAIN_MESSAGE；icon 为在对话框中显示的图标；options 用于指示可能选择的对象组成的数组；如果对象是组件，则可以正确呈现，非 String 对象使用 toString 方法呈现；如果此参数为 null，则由外观确定选项；initialValue 用于表示对话框的默认选择对象，只有在使用 options 时才有意义，可以为 null
public static Object showInputDialog (Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)	selectionValues 用于给出可能选择的 Object 数组；initialSelectionValue 用于初始化输入字段的值

下面将列举一个实例。该实例主要是创建 4 个按钮组件，每个组件的动作事件会显示一个不同类型的对话框。其具体代码如下所示：

```
// 这段程序代码主要是向读者展示 4 种不同的对话框的创建方法
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class test1 extends JPanel
{
    static final int WIDTH=700;
    static final int HEIGHT=400;
    test1()
    {
        JFrame f=new JFrame("对话框测试窗口");
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        f.setLocation(x,y);
        f.setSize(WIDTH,HEIGHT);
        f.setContentPane(this);
        f.setVisible(true);
        setLayout(new FlowLayout());
        JButton b1=new JButton("showMessageDialog");
        // 创建 4 个按钮组件，这 4 个按钮组件引发的动作事件用于显示 4 个不同类型的对话框
        JButton b2=new JButton("showConfirmDialog");
        JButton b3=new JButton("showOptionDialog");
        JButton b4=new JButton("showInputDialog");
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        b1.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent Event)
            {
                JOptionPane.showMessageDialog(null,"这是一个消息对话框");
                // 这是一个 showMessageDialog 静态方法显示的对话框
            }
        });
        b2.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent Event)
            {
                JOptionPane.showConfirmDialog(null,"这是一个有关 confirm 的消息",
                    "这是一个有关 confirm 的对话框",JOptionPane.YES_NO_CANCEL_OPTION);
                // 这是一个 showConfirmDialog 静态方法显示的对话框
            }
        });
        b3.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent Event)
            {
                Object[] options={"OK","CANCEL"};
```



```
JOptionPane.showMessageDialog(null,"点击 ok 按钮继续",
    "警告",JOptionPane.DEFAULT_OPTION,JOptionPane.WARNING_MESSAGE,null,options,options[0]);
}
// 这是一个 showOptionDialog 静态方法显示的对话框
});
b4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        Object[] options={"第一","第二","第三"};
        JOptionPane.showInputDialog(null,"请选择一个","输入",JOptionPane.INFORMATION_MESSAGE,
            new ImageIcon("blue.gif"),options,options[0]);
    } // 这是一个 showInputDialog 静态方法显示的对话框
});
}
public static void main(String[] args)
{
    new test1();
}
}
```

上面程序代码的运行结果如图 12.4 所示。

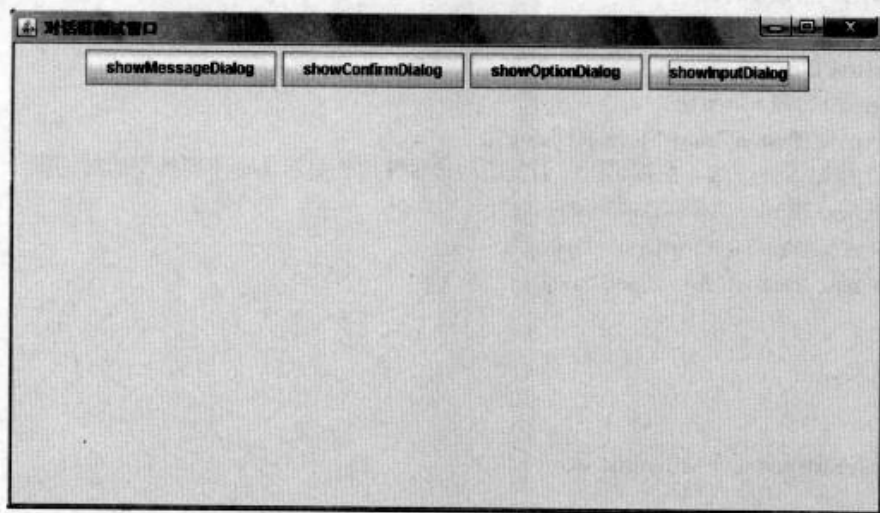


图 12.4 对话框 (a)

当单击 showMessageDialog 按钮时，会弹出如图 12.5 所示的对话框。

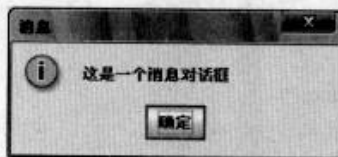


图 12.5 对话框 (b)

当单击 showConfirmDialog 按钮时，会弹出如图 12.6 所示的对话框。

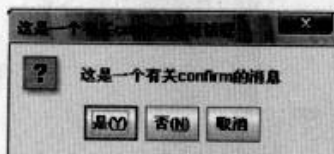


图 12.6 对话框 (c)

当单击 showOptionDialog 按钮时，会弹出如图 12.7 所示的对话框。

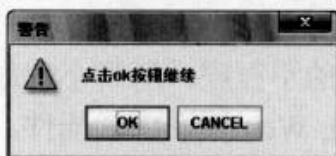


图 12.7 对话框 (d)

当单击 showInputDialog 按钮时，会弹出如图 12.8 所示的对话框。



图 12.8 对话框 (e)

对话框的创建很简单，只不过是一两句代码而已，但是对话框上的每个按钮和下拉列表框的事件需要给予响应。

当调用 showConfirmDialog 和 showOptionDialog 后，用户单击相应的按钮以作出自己选择时，该方法会返回一个整数值以区别不同的用户动作事件，一般来说，是从 0 开始计算的。例如选项类型是 YES_NO_CANCEL_OPTION，那么 0 表示用户单击 YES，如果是 1，表示用户选择的是 NO，选择 2 时表示用户选择的是 CANCEL，所以通常编写程序代码时，是按照返回值的不同以决定程序走向不同的分支、进行不同的操作。

对于 showInputDialog 对话框，则返回一个字符串，该字符串是用户输入的信息。在实际开发中，针对上面的事件处理，是通过条件语句或开关语句来编写的，例如以下程序段：

```
Object[] options={"第一","第二","第三"};
JOptionPane.showInputDialog(null,"请选择一个","输入",
    JOptionPane.INFORMATION_MESSAGE,new ImageIcon("blue.gif"),options,options[0]);
```

为了响应用户单击不同的键，可以使用以下条件语句来处理其事件：

```
if(inuser.equals("第一"))// inuser 是指用户选择的值
{
    ...
}
else if(inuser.equals("第二"))
{
    ...
}
else if(inuser.equals("第三"))
{
    ...
}
else
{
    ...
}
```

采用这种方式可以很好地处理对话框中各个组件的事件。对于对话框的知识，还有很多，限于篇幅问题，在这里仅介绍以上的知识，有兴趣的读者可以自行查阅相关资料。

12.3 如何使用 JApplet 组件

Applet 是运行在 Web 浏览器中的小程序。所谓小程序，就是指那些可以快速运行的网页页面，往往这些页面都是一些使用在 Web 页面中的插件。另外，由于它们应用于互联网络，必须保证其安全，所以 Applet 只能完成有限的功能。但是，Applet 是客户端编程的一个强有力的工具。Applet 编程由于安全原因受到功能方面的限制。Java 运行环境总是监控着 Applet 的所有动作。Applet 的主要目的是扩展 Web 浏览器的功能。为了保证安全，必须有如下限制：

- Applet 不能触及本地计算机的硬盘，也就是说，Applet 不能够读写硬盘，这是因为一般不容许别人利用 Applet 在未经同意的前提下访问私有数据。
- 反对 Applet 直接写入本地硬盘，否则计算机病毒就会很容易侵入。
- 每次运行时，需要将程序从服务器下载到本地计算机，所以 Applet 往往需要较长的时间下载、启动。Web 浏览器可能会缓存部分 Applet 程序，但是并不确保会缓存每个具体的 Applet。正因为如此，应该将 Applet 的各个构件（包括所有的 class 文件、声音文件、图像文件等）压缩到一个“jar”文件，便于将相关文件一次性从服务器下载下来，从而减少连接服务器的次数。

通过继承 javax.swing.JApplet 来实现 Applet 编程。JApplet 定义了一组方法，控制 Applet 的生成以及运行，其方法如表 12.8 所示。

表 12.8 JApplet 所定义的方法

方法名称	用法
init()	Web 浏览器自动调用该方法，完成 Applet 的初始化
start()	每当 Applet 进入 Web 浏览器的可视区域时，被自动调用
stop()	每当 Applet 离开 Web 浏览器的可视区域时，被自动调用
destroy()	当 Applet 终止时，被自动调用

在 Applet 程序中，可以不需要主运行函数，也可以放置在主运行函数里运行。当不需要主运行函数时，就直接将 JApplet 程序放置在 HTML 文档中运行，它是使用 init、start、stop 和 destroy 方法来表示一个程序的运行规律。当需要主运行函数时，则将 init、start、stop 和 destroy 方法作为普通方法放在其中运行即可。

下面将先观察一下不需要主运行函数的情况，其详细的程序代码如下所示：

```
// 这段程序代码主要是为读者展示一个不需要主运行函数的 JApplet 是如何被创建的
import java.JApplet.*;
public class test2 extends JApplet
{
    public void init()                                // 在 init 方法中输出一段话，在初始化页面时会输出它
    {
        System.out.println("欢迎大家进入 JApplet 程序学习页面!");
    }
    public void start()                                // 在 start 方法中输出一段话，在运行时会输出它
    {
        System.out.println("现在程序正在运行。");
    }
    public void stop()
```



```

    {
        System.out.println("程序停止运行了。");
    }
}

```

// 在 stop 方法中输出一段话，在暂停时会输出它

在同一个目录下创建一个 HTML 文件：

```

<html>
<body>
    <Applet code=test.class width="200" height="200">
    </Applet>
</body>
</html>

```

运行结果如下：

```

欢迎大家进入 JApplet 程序学习页面!
现在程序正在运行。
(当关闭 Applet 程序时)
程序停止运行了。

```

上面的实例主要讲述了如何不使用主运行函数来运行 JApplet 程序的方法。下面将介绍如何使用主运行函数来运行独立的 JApplet 程序。独立的应用程序必须包含一个 main() 方法，而在 main() 中生成一个 JFrame 对象，令它作为 Applet 的运行环境即可。

下面给出一个实例，其实例代码如下所示：

```

// 这段程序代码主要向读者展示如何在主运行函数中创建一个 JApplet
import javax.swing.JApplet;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class test3 extends JApplet
{
    public void init()
    {
        getContentPane().add(new JLabel("Applet!"));
    }
    // 应用程序入口：
    public static void main(String[] args)
    {
        JApplet applet = new test3();
        JFrame frame = new JFrame("Applet1c");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(applet);
        frame.setSize(200,80);
        applet.init();
        applet.start();
        frame.setVisible(true);
    }
}

```

上面程序的运行结果如图 12.9 所示。



图 12.9 独立运行的 JApplet 程序

在 `main()` 方法中生成了 `JFrame` 对象 `frame`、`JApplet` 对象 `applet`，然后将 `applet` 添加到 `frame` 的 `ContentPane` 上。通过调用 `applet.init()` 和 `applet.start()` 方法启动这个 `applet`。

在实际开发中，经常会将布局管理器加入到 `JApplet` 中。在 `init()` 方法中调用了 `setLayout()` 方法，这个方法用于设置 `JApplet` 的 `ContentPane` 布局控制器。在这里将其设置为 `FlowLayout`。`JApplet` 默认的布局控制器为 `BorderLayout`，它将 `ContentPane` 分割为 5 个区域。

限于篇幅问题，这里不再赘述，希望读者能够沿着作者的思路自行举例并实践练习。

12.4 本章小结

本章主要是讲述窗口、对话框和 `JApplet` 等顶层容器类的知识。这些容器类都是可以独立存在，无须任何载体，并且是程序的顶层窗口或容器。这些类是开发 `Swing` 图形界面程序的最基本、最重要的类。希望读者能够多多练习，熟练生巧。

12.5 本章习题

1. 设计一个程序，在这个程序中创建一个顶层容器，在这个顶层容器中有 4 个按钮，当单击每个按钮时，都会改变背景颜色。

要求：其最终效果如图 12.10 所示。

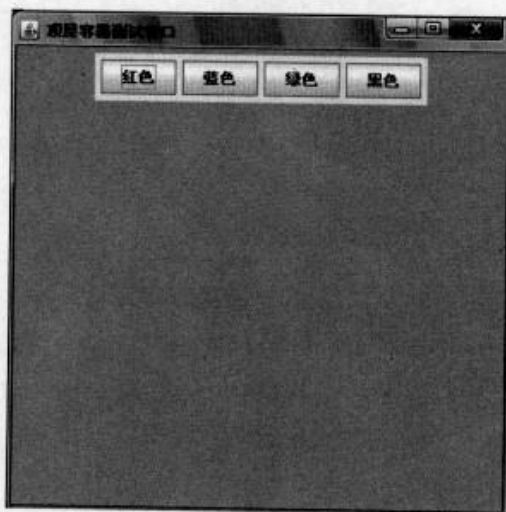


图 12.10 本章习题 1

2. 设计一个程序，在这个程序中有一个顶层容器，在容器中有一个 `JApplet` 程序，另外，在界面中还有 4 个按钮组件，使用这 4 个按钮组件来控制 `JApplet` 程序的运行。

要求：其最终效果如图 12.11 所示。

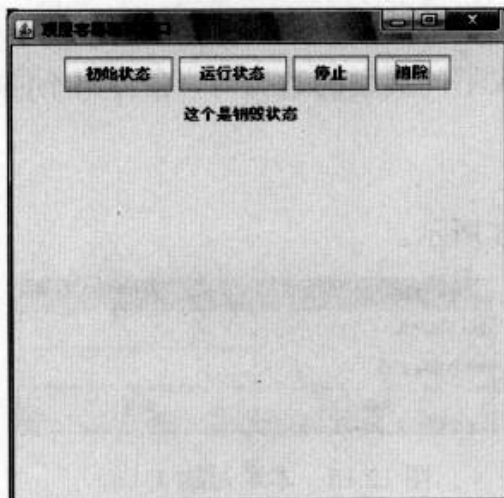


图 12.11 本章习题 2

3. 设计一个程序，在这个程序中创建一个菜单，每个菜单事件都会弹出一个对话框，对话框将控制菜单的其他项是否可用。

要求：

(1) 其最终效果如图 12.12 所示。

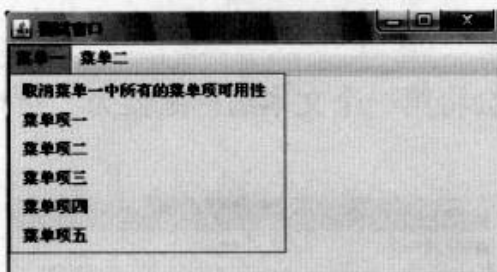


图 12.12 本章习题 3 (a)

(2) 当选择“菜单”|“取消菜单一中所有的菜单项可用性”命令时，会弹出如图 12.13 所示的对话框。

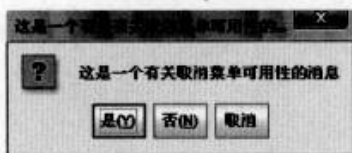


图 12.13 本章习题 3 (b)

(3) 当单击“是”按钮后，则会使菜单中的所有菜单项不可用，其效果如图 12.14 所示。

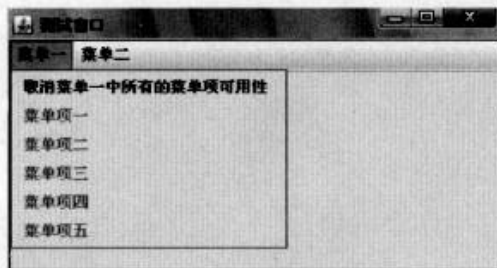


图 12.14 本章习题 3 (c)

4. 设计一个程序, 在这个程序中创建一个顶层容器, 其中有两个标签组件和两个文本组件, 形成一个验证窗口, 当输入不同类型的数据时, 会弹出不同的对话框来提示用户输入的数据是否规范。

要求:

(1) 其最终效果如图 12.15 所示。

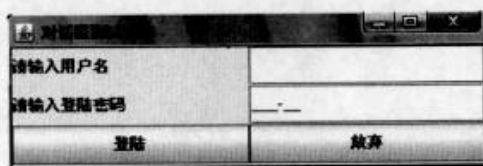


图 12.15 本章习题 4 (a)

(2) 当输入正确的用户名和密码并且单击“登录”按钮时, 会弹出一个对话框, 如图 12.16 所示。

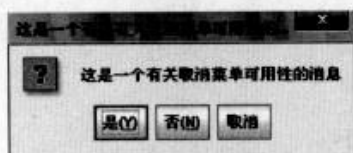


图 12.16 本章习题 4 (b)

(3) 单击“是”按钮后, 会将第一个文本组件赋值为“*****”, 第二个文本组件清空。其效果如图 12.17 所示。

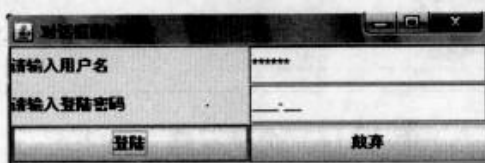


图 12.17 本章习题 4 (c)

第13章 如何使用菜单和工具条组件

在操作一些常见的软件，例如 Word 时，在软件界面上，主要提供给用户操作的就是菜单和工具条。这种方式能够使一个软件的操作简捷、方便和界面更加友好。位于窗口的菜单栏包括了下拉菜单的名字，当单击该名字时，就会打开包含其内的菜单项和子菜单项的菜单。每个菜单项或者子菜单项都会具有相应的事件处理代码，当用户单击某一个菜单项或者子菜单项时，该事件就会发生。而工具条则相当于把菜单中的菜单项或者子菜单项全部以图标的形式平铺在用户面前，以方便其操作，而不需要用户在众多的菜单项中去选择。本章将为读者详细介绍如何创建和使用菜单组件和工具条组件。

13.1 如何使用菜单组件

在本章的章首中提到了工具条就是菜单项或者子菜单项的图像化的平铺，能够方便用户操作，而不需要频繁的在菜单中选择。这样的话，岂不是可以不需要菜单了吗？其实如果将所有的菜单项都以工具条的形式平铺，那么，整个软件将会由大量的图形化的工具项组成。当需要找某个工具的时候，就会在众多的工具中进行选择。由于菜单是按照相似功能进行组合的，所以，用户可以按照相似的功能在菜单中进行选择。由此可知，菜单项和工具条各有各的特性。下面将为读者详细介绍如何创建菜单。

13.1.1 菜单组件的类层次

整个菜单组件分为三个部分：菜单条(JMenuBar)、菜单(JMenu)和菜单项(JMenuItem)。菜单条的类层次如图 13.1 所示。

从上面的层次图中可以看出，JMenuBar 菜单条是不具备 Windows 特性的，所以它也不具备顶层容器类的特性，即不能独立显示出来，必须要依赖于顶层容器才能显示。JMenuBar 组件用来摆入 JMenu 组件。当建立许多 JMenu 组件后，需要通过 JMenuBar 组件来将 JMenu 组件加入到窗口中。菜单的类层次如图 13.2 所示。

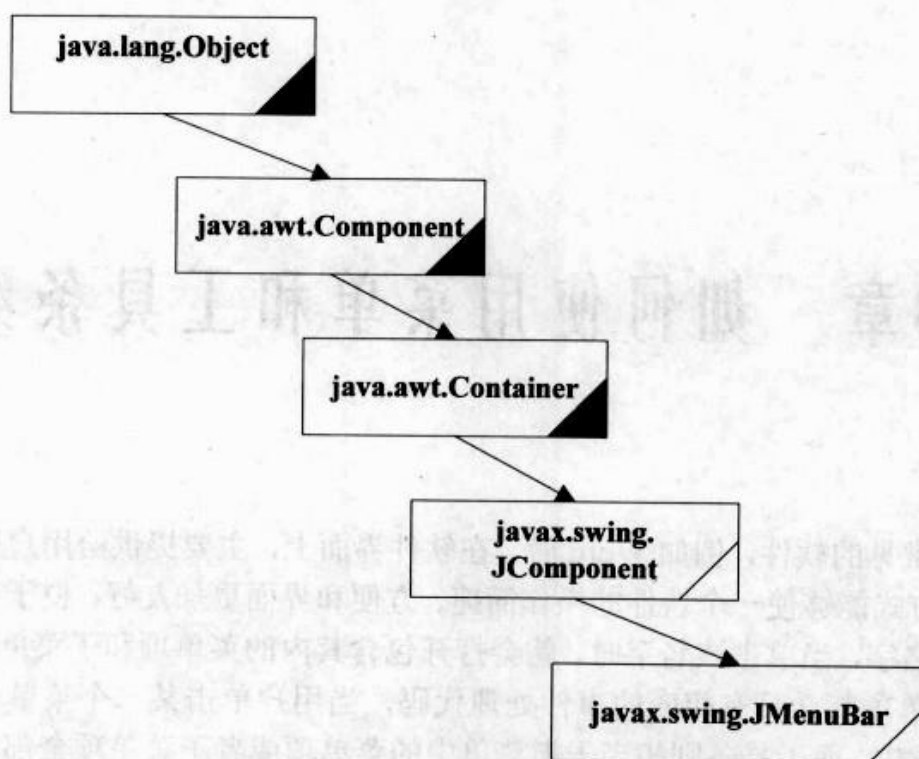


图 13.1 菜单条的类层次图

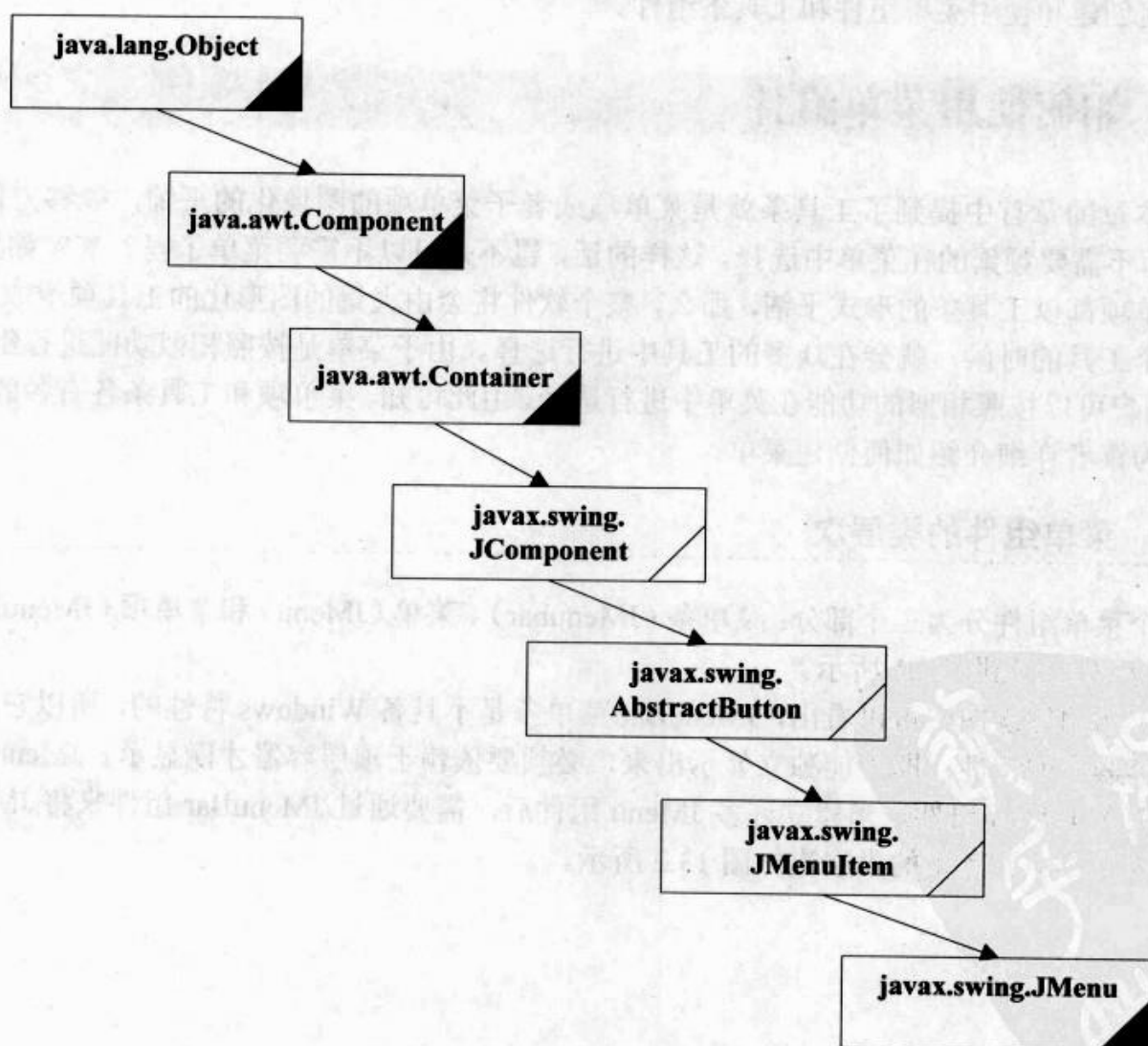


图 13.2 菜单的类层次图

由以上的层次图可知, JMenu 组件具有抽象按钮组件的特性, 即可以通过单击触发事件。JMenu 组件是用来存放和整合 JMenuItem 的组件, 这个组件也是在构成一个菜单中不可缺少的组件之一。JMenu 可以是单一层次的结构, 也可以是一个层次式的结构, 要使用何种形式的结构取决于界面设计上的需要。JMenuItem (菜单项) 的类层次如图 13.3 所示。

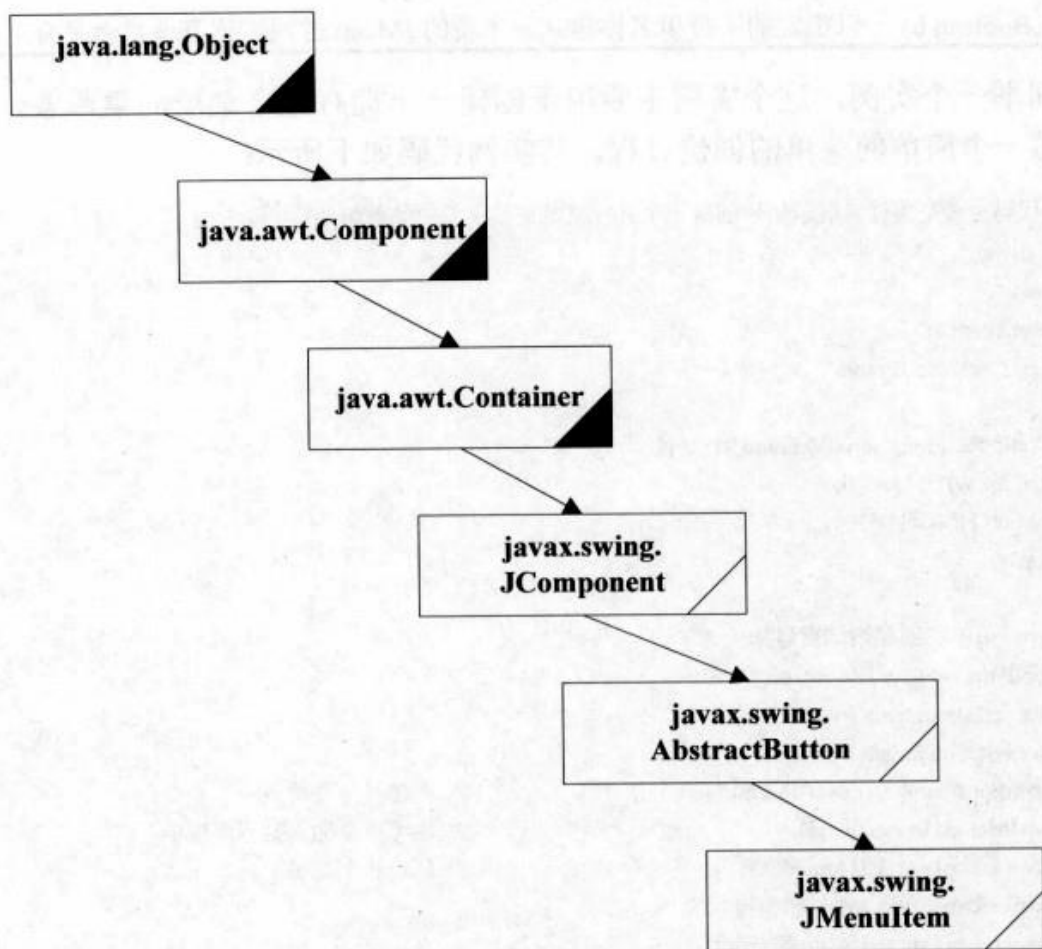


图 13.3 菜单项的类层次图

由以上层次图可知, 同 JMenu 组件一样, JMenuItem 是一种特殊的按钮组件, 所以 JMenuItem 支持许多在按钮组件中惯用的功能, 例如加入图标文件或是在菜单中选择某一项菜单项时就如同按下按钮的操作一样触发 `ActionEvent`, 通过动作事件的机制就能针对不同的 JMenuItem 编写其代码。

13.1.2 如何创建菜单

正如前面所述, 创建一个菜单, 要分成三个步骤:

- 01 先创建一个菜单条, 用来存放菜单。
- 02 创建菜单, 用来存放菜单项。
- 03 创建菜单项, 也就是用户需要的功能。

下面将详细讲述其创建的过程。要创建一个菜单条, 可以通过其构造器来创建。菜单条的构造器非常简单, 如 `JMenuBar()`, 用于建立一个新的 `JMenuBar`。由于构造一个空的 `JMenuBar` 后设置到窗口上, 对于窗口来说是没有意义的, 因此 `JMenuBar` 需要结合至少一个以上的 `JMenu` 组件才会在画面上显现出视觉的效果。JMenu 的构造器说明如表 13.1 所示。

表 13.1 JMenu 构造器

JMenu 构造器	说明
JMenu()	建立一个新的 JMenu
JMenu(String s)	以指定的字符串名称建立一个新的 JMenu
JMenu(String s,Boolean b)	以指定的字符串名称建立一个新的 JMenu 并决定该菜单是否具有下拉式的属性

下面将列举一个实例，这个实例主要用于创建一个拥有几个菜单的菜单条，通过实例，读者可以熟悉一个简单的菜单的创建过程。其实例代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建一个具有菜单条和菜单的菜单组件
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test1 extends JFrame
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public test1()
    {
        super("菜单的创建测试窗口");
        JRootPane rp=new JRootPane();
        super.setContentPane(rp);
        super.setContentPane(rp);
        JMenuBar menubar1=new JMenuBar();           // 创建一个菜单条
        rp.setJMenuBar(menubar1);                   // 将菜单条加入到根面板中
        JMenu menu1=new JMenu("文件");              // 创建几个菜单
        JMenu menu2=new JMenu("编辑");
        JMenu menu3=new JMenu("视图");
        JMenu menu4=new JMenu("运行");
        JMenu menu5=new JMenu("工具");
        JMenu menu6=new JMenu("帮助");
        menubar1.add(menu1);                         // 在菜单条中添加菜单
        menubar1.add(menu2);
        menubar1.add(menu3);
        menubar1.add(menu4);
        menubar1.add(menu5);
        menubar1.add(menu6);
        this.setVisible(true);
    }
    public static void main(String args[])
    {
        new test1();
    }
}
```

上面程序代码的运行结果如图 13.4 所示。

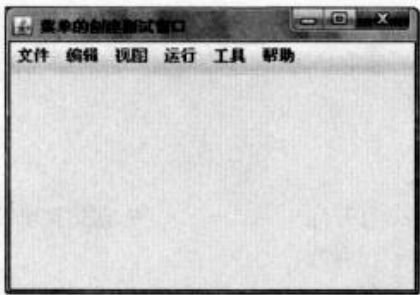


图 13.4 不带菜单项的菜单创建

有了菜单，虽然可以形成简单的菜单程序，但是这种菜单不专业，而这个程序也没有必要做成菜单。真正专业的菜单是需要菜单项的，所以接下来将介绍如何创建菜单项组件，菜单项组件的构造器说明如表 13.2 所示。

表 13.2 JMenuItem 构造器

JMenuItem 构造器	说明
JMenuItem()	建立一个新的 JMenuItem
JMenuItem(Icon icon)	建立一个有图标的 JMenuItem
JMenuItem(String text)	建立一个有文字的 JMenuItem
JMenuItem(String text,Icon icon)	建立一个有图标和文字的 JMenuItem
JMenuItem(String text,int mnemonic)	建立一个有文字和键盘设置快捷键的 JMenuItem

上述表格给出了菜单项不同类型的构造器，在实际开发中最常用的构造器是 JMenuItem(String text)和 JMenuItem(String text,Icon icon)两种。下面将给出一个实例，通过实例让读者熟悉如何创建菜单。其实例程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何在菜单中添加菜单项
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test2 extends JFrame
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public test2()
    {
        super("测试窗口");
        JRootPane rp=new JRootPane();
        super.setContentPane(rp);
        super.setContentPane(rp);
        JMenuBar menubar1=new JMenuBar();
        rp.setJMenuBar(menubar1);
        JMenu menu1=new JMenu("文件");
        JMenu menu2=new JMenu("编辑");
        JMenu menu3=new JMenu("视图");
        JMenu menu4=new JMenu("运行");
        JMenu menu5=new JMenu("工具");
        JMenu menu6=new JMenu("帮助");
        menubar1.add(menu1);
```



```
menubar1.add(menu2);
menubar1.add(menu3);
menubar1.add(menu4);
menubar1.add(menu5);
menubar1.add(menu6);
JMenuItem item1=new JMenuItem("打开");           // 创建菜单项组件
JMenuItem item2=new JMenuItem("保存");
JMenuItem item3=new JMenuItem("打印");
JMenuItem item4=new JMenuItem("退出");
JMenuItem item5=new JMenuItem("查找");
JMenuItem item6=new JMenuItem("替换");
JMenuItem item7=new JMenuItem("剪切");
JMenuItem item8=new JMenuItem("拷贝");
JMenuItem item9=new JMenuItem("展开图");
JMenuItem item10=new JMenuItem("分屏图");
JMenuItem item11=new JMenuItem("在 dos 下运行");
JMenuItem item12=new JMenuItem("在 windows 下运行");
JMenuItem item13=new JMenuItem("看图工具");
JMenuItem item14=new JMenuItem("快速运行工具");
JMenuItem item15=new JMenuItem("版本号");
JMenuItem item16=new JMenuItem("帮助信息");
menu1.add(item1);                                   // 在不同的菜单组件中添加不同的菜单项
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu1.addSeparator();
menu1.add(item4);
menu2.add(item5);
menu2.add(item6);
menu2.addSeparator();
menu2.add(item7);
menu2.add(item8);
menu3.add(item9);
menu3.add(item10);
menu4.add(item11);
menu4.add(item12);
menu5.add(item13);
menu5.add(item14);
menu6.add(item15);
menu6.add(item16);
this.setVisible(true);
this.pack();
}
public static void main(String args[])
{
    new test2();
}
```

上面程序代码的运行结果如图 13.5 所示。

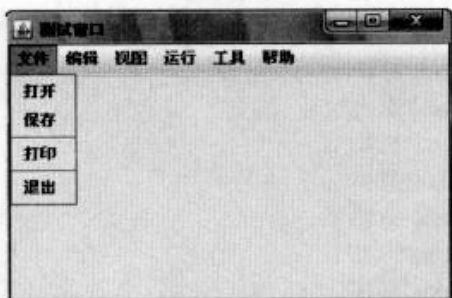


图 13.5 带菜单项的菜单的创建

以上介绍了整个菜单的创建方法。但是每个菜单项的功能都是通过一个动作事件来处理的，下小节将介绍如何实现菜单项的功能。

13.1.3 如何处理菜单事件

本小节将主要讲述如何通过动作事件处理菜单项的事件响应。菜单项的事件处理机制类似于按钮组件的事件处理模式，当按下 `JMenuItem` 组件时，就如同按下 `JButton` 组件一般，均会产生 `ActionEvent` 事件。

下面将列举一个实例，这个实例主要是实现一个菜单项的动作事件处理。通过实例使读者熟悉处理菜单的事件过程。其实例程序代码如下所示：

// 这段程序代码主要介绍一个菜单的动作事件的处理方法

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class test3
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JFrame f;
    public test3()
    {
        f=new JFrame("测试窗口");
        JRootPane rp=new JRootPane();
        f.setContentPane(rp);
        f.setContentPane(rp);
        JMenuBar menubar1=new JMenuBar();
        rp.setJMenuBar(menubar1);
        JMenu menu1=new JMenu("文件");
        JMenu menu2=new JMenu("编辑");
        JMenu menu3=new JMenu("视图");
        JMenu menu4=new JMenu("运行");
        JMenu menu5=new JMenu("工具");
        JMenu menu6=new JMenu("帮助");
        menubar1.add(menu1);
        menubar1.add(menu2);
        menubar1.add(menu3);
        menubar1.add(menu4);
        menubar1.add(menu5);
        menubar1.add(menu6);
        JMenuItem item1=new JMenuItem("打开");
```



```

JMenuItem item2=new JMenuItem("保存");
JMenuItem item3=new JMenuItem("打印");
JMenuItem item4=new JMenuItem("退出");
JMenuItem item5=new JMenuItem("查找");
JMenuItem item6=new JMenuItem("替换");
JMenuItem item7=new JMenuItem("剪切");
JMenuItem item8=new JMenuItem("拷贝");
JMenuItem item9=new JMenuItem("展开图");
JMenuItem item10=new JMenuItem("分屏图");
JMenuItem item11=new JMenuItem("在 dos 下运行");
JMenuItem item12=new JMenuItem("在 windows 下运行");
JMenuItem item13=new JMenuItem("看图工具");
JMenuItem item14=new JMenuItem("快速运行工具");
JMenuItem item15=new JMenuItem("版本信息");
JMenuItem item16=new JMenuItem("帮助信息");
menu1.add(item1);
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu1.addSeparator();
menu1.add(item4);
menu2.add(item5);
menu2.add(item6);
menu2.addSeparator();
menu2.add(item7);
menu2.add(item8);
menu3.add(item9);
menu3.add(item10);
menu4.add(item11);
menu4.add(item12);
menu5.add(item13);
menu5.add(item14);
menu6.add(item15);
menu6.add(item16);
f.setVisible(true);
f.setSize(WIDTH,HEIGHT);
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);
/* 以上的程序段主要是创建了菜单和菜单项的界面, 下面为这些菜单项的动作事件处理*/
// 处理退出菜单项的动作事件
item4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
            "退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        // 通过对话框中按钮的选择来决定结果, 单击 yes 按钮时, 窗口直接消失
        if(i==0)
            f.dispose();
    }
}

```



```
});  
}  
public static void main(String args[])  
{  
    new test2();  
}  
}
```

上面程序代码的运行结果如图 13.6 所示。

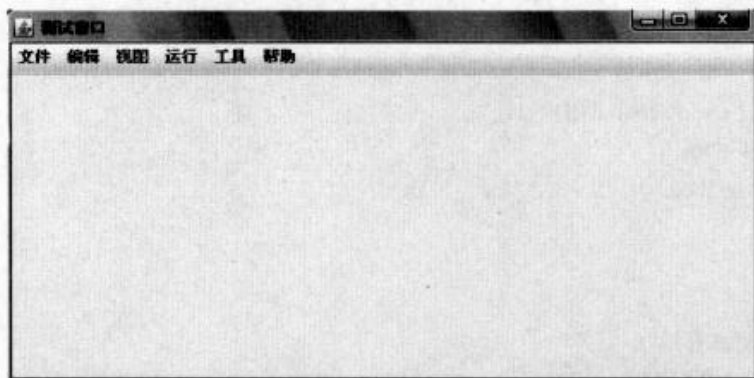


图 13.6 菜单项事件处理

当选择“文件”|“退出”命令时，会弹出如图 13.7 所示的对话框。

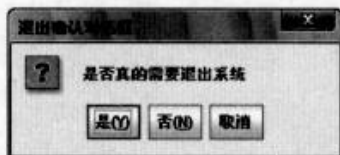


图 13.7 退出的事件处理

其实菜单项的事件处理没有什么特殊的地方，与其他组件的动作事件处理的方式一样，这里不再赘述。

13.1.4 如何响应键盘操作

读者是否有过这样的经验：在很多菜单项的旁边有快捷键，用户可以通过键盘的操作达到同选择菜单一样的效果，那么如何来响应键盘的操作呢？本小节将围绕这个知识点展开讨论。

通常很多用户比较习惯使用菜单的快捷键来操作，因为这样对于熟悉软件的用户来说，非常方便和快捷。在 Java 图形程序开发中，可以通过在菜单项的构造器中指定一个字母作为菜单项的快捷键。其设置方法如下所示：

```
JMenuItem copyItem=new JMenuItem("copy",'c');
```

而对于菜单来说，也可以设置其快捷键，不过它的设置方法与菜单项的设置方式不一样。它的方法如下所示：

```
JMenu Filemenu=new JMenu("文件");  
Filemenu.setMnemonic('F');
```


通过这种方式设置菜单后,如果要使用菜单快捷键,就可以使用“Alt+快捷键”来操作菜单。下面将列举个实例,该实例为某些菜单和菜单项设置了快捷键。通过实例可使读者熟悉使用键盘快捷键来操作菜单和菜单项的方法。其程序代码如下所示:

// 这段程序代码主要是为读者展示如何为菜单或者菜单项设置快捷键

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test3
```

```
{
```

```
    private static final long serialVersionUID = 1L;
```

```
    static final int WIDTH=600;
```

```
    static final int HEIGHT=300;
```

```
    JFrame f;
```

```
    public test3()
```

```
    {
```

```
        f=new JFrame("测试窗口");
```

```
        JRootPane rp=new JRootPane();
```

```
        f.setContentPane(rp);
```

```
        f.setContentPane(rp);
```

```
        JMenuBar menubar1=new JMenuBar();
```

```
        rp.setJMenuBar(menubar1);
```

```
        JMenu menu1=new JMenu("文件");
```

```
        JMenu menu2=new JMenu("编辑");
```

```
        JMenu menu3=new JMenu("视图");
```

```
        JMenu menu4=new JMenu("运行");
```

```
        JMenu menu5=new JMenu("工具");
```

```
        JMenu menu6=new JMenu("帮助");
```

```
        menu1.setMnemonic('F');
```

// 设置了几个菜单的快捷键

```
        menu2.setMnemonic('E');
```

```
        menu3.setMnemonic('V');
```

```
        menu4.setMnemonic('R');
```

```
        menu5.setMnemonic('T');
```

```
        menu6.setMnemonic('H');
```

```
        menubar1.add(menu1);
```

```
        menubar1.add(menu2);
```

```
        menubar1.add(menu3);
```

```
        menubar1.add(menu4);
```

```
        menubar1.add(menu5);
```

```
        menubar1.add(menu6);
```

```
        JMenuItem item1=new JMenuItem("打开",'O');
```

// 设置了几个菜单项的快捷键

```
        JMenuItem item2=new JMenuItem("保存",'S');
```

```
        JMenuItem item3=new JMenuItem("打印",'P');
```

```
        JMenuItem item4=new JMenuItem("退出",'Q');
```

```
        JMenuItem item5=new JMenuItem("查找");
```

```
        JMenuItem item6=new JMenuItem("替换");
```

```
        JMenuItem item7=new JMenuItem("剪切");
```

```
        JMenuItem item8=new JMenuItem("拷贝");
```

```
        JMenuItem item9=new JMenuItem("展开图");
```

```
        JMenuItem item10=new JMenuItem("分屏图");
```

```
        JMenuItem item11=new JMenuItem("在 DOS 下运行");
```



```
JMenuItem item12=new JMenuItem("在 Windows 下运行");
JMenuItem item13=new JMenuItem("看图工具");
JMenuItem item14=new JMenuItem("快速运行工具");
JMenuItem item15=new JMenuItem("版本信息");
JMenuItem item16=new JMenuItem("帮助信息");
menu1.add(item1);
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu1.addSeparator();
menu1.add(item4);
menu2.add(item5);
menu2.add(item6);
menu2.addSeparator();
menu2.add(item7);
menu2.add(item8);
menu3.add(item9);
menu3.add(item10);
menu4.add(item11);
menu4.add(item12);
menu5.add(item13);
menu5.add(item14);
menu6.add(item15);
menu6.add(item16);
f.setVisible(true);
f.setSize(WIDTH,HEIGHT);
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);
item4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
            "退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            f.dispose();
        }
    }
});
}
public static void main(String args[])
{
    new test3();
}
```

上面程序代码的运行结果如图 13.8 所示。

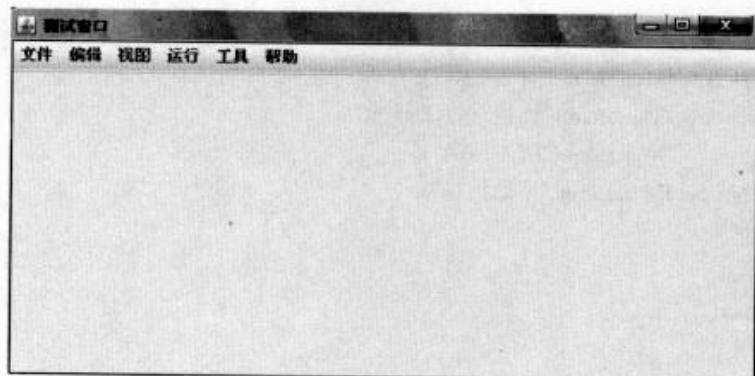


图 13.8 快捷键的设置 (a)

当单击快捷键 Alt+F 后, “文件” 菜单就会被打开, 如图 13.9 所示。

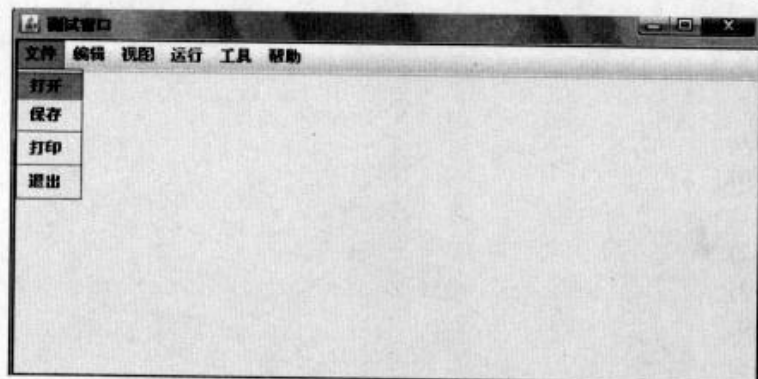


图 13.9 快捷键的设置 (b)

单击快捷键 Q 时, 系统会弹出退出对话框, 如图 13.10 所示。

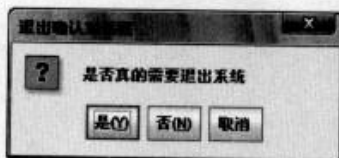


图 13.10 快捷键的设置 (c)

需要注意的是: 菜单和菜单项的快捷键设置是不同的, 菜单项是采用其本身的构造器来进行设置, 而菜单是利用其类中的方法实现的。

13.1.5 如何使用弹出式菜单

在实际的应用中经常会遇到弹出式菜单, 什么是弹出式菜单? 其实就是指平时大家所说的右键菜单, 而这个菜单就是把使用频率比较高的一些菜单项放入其中, 从而能够方便用户快捷地操作软件。在本小节中将围绕着这个知识点来进行讨论。

前面讲述了固定菜单的创建和使用的方法, 但是有的菜单项是在用户平时使用时非常频繁的, 此时, 可以使用弹出式菜单。弹出式菜单把所有用户经常使用的菜单项放到一个菜单中, 这样, 当用户右键鼠标时, 就可以很容易地找到自己需要的菜单项。

弹出式菜单是通过 `JPopupMenu` 类实现的, 但弹出式菜单是没有标题的, 其创建方法如下:

```
JPopupMenu popmenu=new JPopupMenu();
```

创建好了弹出式菜单后, 就可以在其中添加所需要的菜单项, 其添加方式如下所示:


```
popmenu.add(菜单项);
```

下面将给出一个实例，这个实例先创建一个固定的常规菜单组件，然后将一些常用的菜单项添加到弹出式菜单中。通过实例可使读者熟悉创建弹出式菜单的创建方法，其程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建一个弹出式菜单
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test5
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JPopupMenu pop;
    JFrame f;
    public test5()
    {
        ...
        // 与上述内容中相同实例的代码相似，这里不再列出
        Container con=new Container();
        f.setContentPane(con);
        pop=new JPopupMenu();           // 创建一个弹出式菜单对象
        pop.add(item4);                 // 将一些菜单项添加到弹出式菜单中
        pop.addSeparator();
        pop.add(item5);
        pop.addSeparator();
        pop.add(item8);
        pop.addSeparator();
        pop.add(item16);
        ...
        // 与上述内容中相同实例的代码相似，这里不再列出
    }
    con.addMouseListener(new MouseAdapter()
    {
        public void mouseReleased(MouseEvent event)
        {
            if(event.isPopupTrigger())
                pop.show(event.getComponent(),event.getX(),event.getY());
        }
    });
}
public static void main(String args[])
{
    new test5();
}
```

上面程序代码的运行结果如图 13.11 所示。

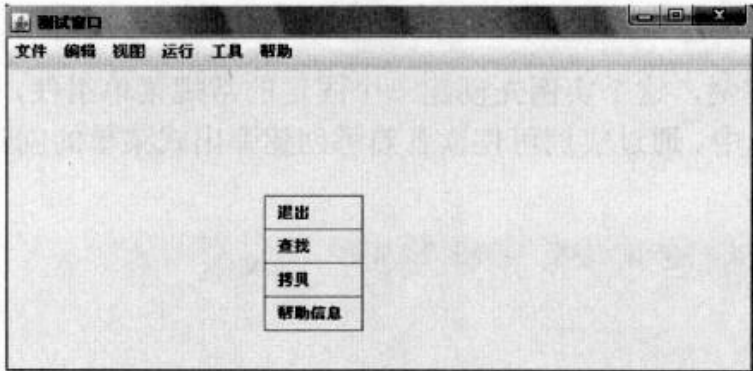


图 13.11 弹出式菜单的创建

通过上面的程序实例可以总结一下创建弹出式菜单的步骤：

- 01

使用 JPopupMenu popmenu=new JPopupMenu()创建一个空的弹出式菜单。
- 02

使用 popmenu.add(菜单项)方式将菜单项添加到弹出式菜单中。
- 03

在内容面板中添加一个鼠标右键事件的处理代码，代码如下：

```
con.addMouseListener(new MouseAdapter()
{
    public void mouseReleased(MouseEvent event)
    {
        if(event.isPopupTrigger())
            pop.show(event.getComponent(),event.getX(),event.getY());
    }
})
```

这样就可以在窗口中的任意空白处使用右键实现弹出式菜单的功能。通过上面的讲述，读者可以根据这几个步骤来创建弹出式菜单。

13.1.6 如何使用菜单项的启用和禁用功能

在平时使用软件的时候，可能会发现一个现象，就是当选择某个菜单项时，另一个菜单项就会不可用，或者当选择一个菜单项时其他的菜单项才能使用。这种现象就是本小节将要讲述的菜单项的启用和禁用功能。

如果要想实现上面所述的功能，需要进行以下两个操作：

- 01

将需要实现功能的菜单添加到菜单监听器。
- 02

在菜单监听器中实现 setEnabled()方法。

在菜单监听器接口中声明了三个方法，这些方法以及这些方法的用法如表 13.3 所示。

表 13.3 菜单监听器接口中的方法

菜单监听器接口中的方法	说明
public void menuSelected(MenuEvent event)	选择某个菜单时调用
public void menuDeselected(MenuEvent event)	取消选择某个菜单时调用
public void menuCanceled(MenuEvent event)	取消菜单时调用

所以要实现菜单监听，就必须要实现上述接口中的三个方法。为了能够实现启用和禁用功能，一般要实现 menuSelected()方法。在程序代码中一般会使用如下代码来实现菜单项的启用和禁用的功能：

```
菜单.addMenuListener(new MenuListener()
{
    public void menuSelected(MenuEvent event)
    {
        菜单项 1.setEnabled(!菜单项 2.isSelected());
    }
    ...
})
```

这里的菜单项 1 就是指需要启用和禁用的菜单项，而这里的菜单项 2 就是让菜单项 1 能够启用和禁用的菜单项。在后面的小节中会通过实例让读者有一个清晰的概念。

13.1.7 如何创建复选框菜单项

复选框菜单项就是在菜单文本旁边显示的一个复选框。当用户选择该菜单时，该菜单会在选中和未选中之间进行切换。复选框菜单项的构造器的说明如表 13.4 所示。

表 13.4 复选框菜单项构造器

复选框菜单项构造器	说明
JCheckBoxMenuItem()	建立一个新的复选框菜单项
JCheckBoxMenuItem(Icon icon)	建立一个有图标的复选框菜单项
JCheckBoxMenuItem(String text)	建立一个有文字的复选框菜单项
JCheckBoxMenuItem(String text,Boolean b)	建立一个有文字和设置选择状态的复选框菜单项
JCheckBoxMenuItem(String text,Icon icon)	建立一个有文字和图标的复选框菜单项
JCheckBoxMenuItem(String text,Icon icon,Boolean b)	建立一个有文字、图标和设置状态的复选框菜单项

上述表格列举了复选框菜单项的常用构造器，下面将通过实例来讲解创建复选框菜单项的方法。该实例主要是在菜单中创建一个复选框菜单项，其具体的实例代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建一个复选框菜单项
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test6
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JPopupMenu pop;
    JMenuItem item2;
    JFrame f;
    JCheckBoxMenuItem item1;
    public test6()
    {
        Item1=new JCheckBoxMenuItem("打开");
```



```
...
// 与上述内容中相同实例的代码相似, 这里不再列出
item1.setAccelerator( KeyStroke.getKeyStroke('O', java.awt.Event.CTRL_MASK, true) );
item2.setAccelerator( KeyStroke.getKeyStroke('S', java.awt.Event.CTRL_MASK, true) );
item3.setAccelerator( KeyStroke.getKeyStroke('P', java.awt.Event.CTRL_MASK, true) );
item4.setAccelerator( KeyStroke.getKeyStroke('Q', java.awt.Event.CTRL_MASK, true) );
...
// 与上述内容中相同实例的代码相似, 这里不再列出
menu1.addMenuListener(new MenuListener()
{
    // 在菜单鼠标监听器接口中设置当单击打开菜单项时, 保存菜单项将被禁用
    public void menuSelected(MenuEvent event)
    {
        item2.setEnabled(!item1.isSelected());
    }
    public void menuDeselected(MenuEvent event){}
    public void menuCanceled(MenuEvent event){}
});
}
public static void main(String args[])
{
    new test6();
}
```

上面的程序运行后, 选择“文件”命令后, 会出现如图 13.12 所示对话框。

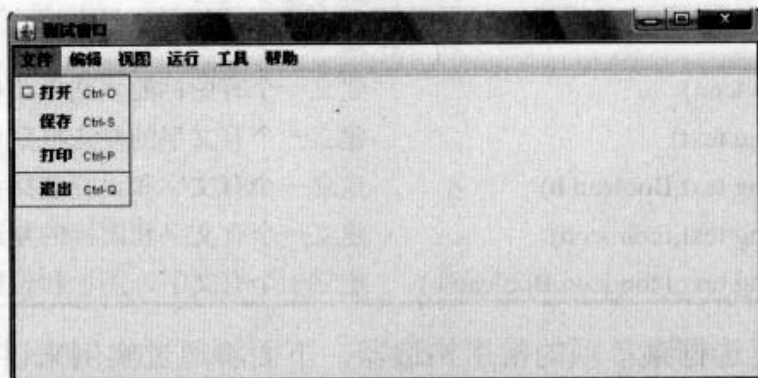


图 13.12 复选框菜单的创建 (a)

当选择“文件”|“打开”命令后, 会出现如图 13.13 所示对话框。

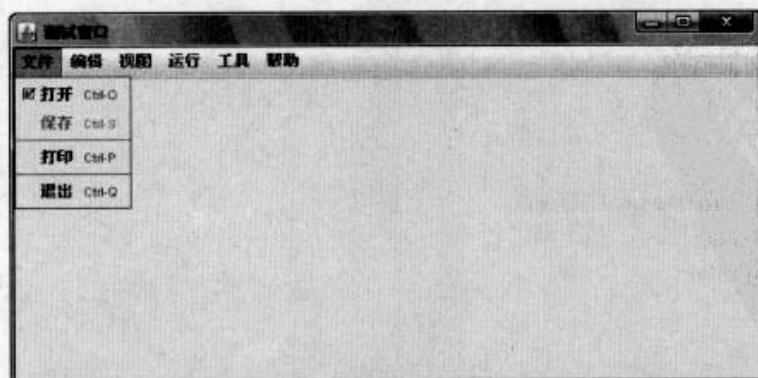


图 13.13 复选框菜单的创建 (二)

可以看出，当选择“打开”命令时，“保存”命令会被禁用。



13.1.8 如何创建单选按钮菜单项

单选按钮菜单项就是在菜单文本旁有一个单选按钮，通过用户选择该菜单项，可以在选中和未选中之间切换。单选按钮菜单项的常用构造器说明如表 13.5 所示。

表 13.5 单选按钮菜单项构造器

单选按钮菜单项构造器	说明
JRadioButtonMenuItem()	建立一个新的单选按钮菜单
JRadioButtonMenuItem(Icon icon)	建立一个有图标的单选按钮菜单
JRadioButtonMenuItem(Icon icon,Boolean selected)	建立一个有图标和设置选择状态的单选按钮菜单
JRadioButtonMenuItem(String text)	建立一个有文字的单选按钮菜单
JRadioButtonMenuItem(String text,Boolean selected)	建立一个有文字和设置选择状态的单选按钮菜单
JRadioButtonMenuItem(String text,Icon icon)	建立一个有文字和图标的单选按钮菜单
JRadioButtonMenuItem(String text,Icon icon,Boolean selected)	建立一个有文字、图标和设置状态的单选按钮菜单

针对上面所介绍的常用构造器设计一个实例，该实例主要用于创建一个单选按钮菜单项，从而使读者能够掌握创建单选按钮菜单项的方法。其具体的程序实例代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建单选按钮菜单项
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test7
{
    JRadioButtonMenuItem item1;
    ...
    // 与上述内容中相同实例的代码相似，这里不再列出
    menu1.addMenuListener(new MenuListener()
    {
        public void menuSelected(MenuEvent event)
        {
            item2.setEnabled(!item1.isSelected());
        }
        public void menuDeselected(MenuEvent event){}
        public void menuCanceled(MenuEvent event){}
    });
    ...
    public static void main(String args[])
    {
        new test7();
    }
}
```

上面的程序代码运行后，选择“文件”命令，会出现如图 13.14 所示的对话框。

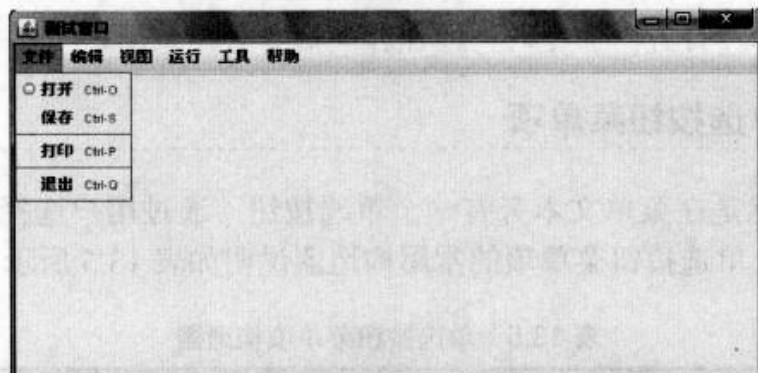


图 13.14 创建单选按钮菜单 (a)

当选择“文件”|“打开”命令时，保存菜单项会被禁用，如图 13.15 所示。

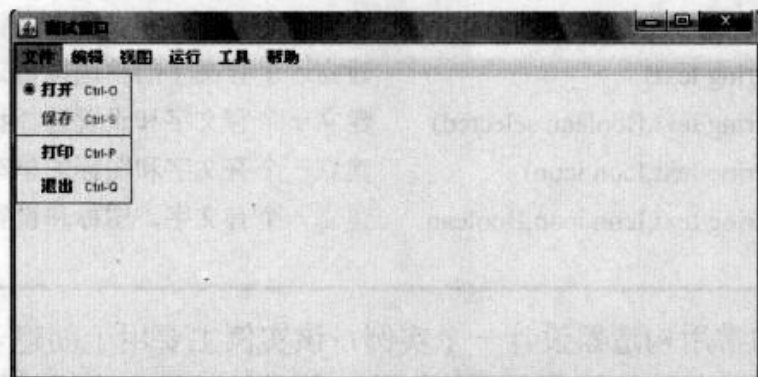


图 13.15 创建单选按钮菜单 (b)

无论是复选框菜单项还是单选按钮菜单项，其使用方法都是一样的，可以实现相同的功能。

13.1.9 如何定义个性化菜单

本小节将利用前面介绍的知识创建一个个性化的菜单。先设计一个实例，程序实例代码如下所示：

```
/* 这段程序主要是创建一个有关“星火纺织厂信息系统”的菜单，并且创建一个 JTabbedPane 组件
* 将有关信息放在其中，当选择菜单中的第一项时，会弹出这个 JTabbedPane 组件，显示出相应的信息
*/
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test8
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JPopupMenu pop;
    JMenuItem item2;
    JFrame f;
    JMenuItem item1;
    JPanel p;
    JTabbedPane tp;
    public test8()
    {
```



```
f=new JFrame("星火纺织厂信息系统");
JMenuBar menubar1=new JMenuBar();
tp=new JTabbedPane();
p=new JPanel();
f.setJMenuBar(menubar1);
f.add(p);
p.add(tp);
JMenu menu1=new JMenu("职工信息系统");
JMenu menu2=new JMenu("中层干部信息系统");
JMenu menu3=new JMenu("工资系统");
JMenu menu4=new JMenu("查询系统");
JMenu menu5=new JMenu("登录系统");
JMenu menu6=new JMenu("帮助系统");
menu1.setMnemonic('Z');
menu2.setMnemonic('C');
menu3.setMnemonic('G');
menu4.setMnemonic('F');
menu5.setMnemonic('D');
menu6.setMnemonic('H');
menubar1.add(menu1);
menubar1.add(menu2);
menubar1.add(menu3);
menubar1.add(menu4);
menubar1.add(menu5);
menubar1.add(menu6);
item1=new JMenuItem("磨砂分厂职工信息");
item2=new JMenuItem("纺纱分厂职工信息");
JMenuItem item3=new JMenuItem("人事部职工信息");
JMenuItem item4=new JMenuItem("财务部职工信息");
JMenuItem item5=new JMenuItem("材料科职工信息");
JMenuItem item6=new JMenuItem("销售科职工信息");
JMenuItem item7=new JMenuItem("成品车间职工信息");
JMenuItem item8=new JMenuItem("分厂厂长信息");
JMenuItem item9=new JMenuItem("部门经理信息");
JMenuItem item10=new JMenuItem("职工工资信息");
JMenuItem item11=new JMenuItem("领导工资信息");
JMenuItem item12=new JMenuItem("按照姓名查询");
JMenuItem item13=new JMenuItem("按照工号查询");
JMenuItem item14=new JMenuItem("添加登录用户名");
JMenuItem item15=new JMenuItem("版本信息");
JMenuItem item16=new JMenuItem("帮助信息");
item1.setAccelerator( KeyStroke.getKeyStroke('M', java.awt.Event.CTRL_MASK, false) );
item2.setAccelerator( KeyStroke.getKeyStroke('F', java.awt.Event.CTRL_MASK, false) );
item3.setAccelerator( KeyStroke.getKeyStroke('R', java.awt.Event.CTRL_MASK, false) );
item4.setAccelerator( KeyStroke.getKeyStroke('C', java.awt.Event.CTRL_MASK, false) );
menu1.add(item1);
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu1.add(item4);
menu1.add(item5);
menu1.add(item6);
menu1.add(item7);
menu2.add(item8);
menu2.add(item9);
```



```

menu3.add(item10);
menu3.add(item11);
menu4.add(item12);
menu4.add(item13);
menu5.add(item14);
menu6.add(item15);
menu6.add(item16);
f.setVisible(true);
f.setSize(WIDTH,HEIGHT);
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);
item1.addActionListener(new ActionListener()
{
    /* 处理菜单项的动作事件，当单击菜单项时，会在中间容器中显示出一个 JTabbedPane 组件
    * 在这个组件中，会有一些排列好的组件
    */
    public void actionPerformed(ActionEvent Event)
    {
        // info 是一个组件排列类
        info con1=new info();
        // 下面是创建一个 JTabbedPane 组件
        JPanel con2 = new JPanel ();
        JPanel con3 = new JPanel ();
        JPanel con4 = new JPanel ();
        JPanel con5 = new JPanel ();
        tp.addTab("con1", con1);
        tp.setEnabledAt(0,true);
        tp.setTitleAt(0,"一车间信息");
        tp.addTab ("con2", con2);
        tp.setEnabledAt (1, true);
        tp.setTitleAt (1,"二车间信息");
        tp.addTab ("con3", con3);
        tp.setEnabledAt (2, true);
        tp.setTitleAt (2,"三车间信息");
        tp.addTab ("con4", con4);
        tp.setEnabledAt(0,true);
        tp.setTitleAt(3,"四车间信息");
        tp.addTab ("con5", con5);
        tp.setEnabledAt(4,true);
        tp.setTitleAt(4,"五车间信息");
        tp.setPreferredSize (new Dimension (500,200));
        tp.setTabPlacement (JTabbedPane.TOP);
        tp.setTabLayoutPolicy (JTabbedPane.SCROLL_TAB_LAYOUT);
    }
});
item4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
            "退出确认对话框", JOptionPane.YES_NO_CANCEL_OPTION);
    }
}

```



```

        if(i==0)
        {
            f.dispose();
        }
    }
});
menu1.addMenuListener(new MenuListener()
{
    public void menuSelected(MenuEvent event)
    {
        item2.setEnabled(!item1.isSelected());
    }
    public void menuDeselected(MenuEvent event){}
    public void menuCanceled(MenuEvent event){}
});
}
public static void main(String args[])
{
    new test8();
}
}

```

在标签容器中放置一些组件，这些组件要经过 GridBagLayout 布局方式布局，所以将其放置在同一个类中，其具体代码如下所示：

class info extends JPanel// 创建一个组件排列布局类

```

{
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    info()
    {
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
        JLabel name=new JLabel("姓名");
        JLabel code=new JLabel("工号");
        JLabel sexy=new JLabel("性别");
        JLabel age=new JLabel("年龄");
        JLabel birthday=new JLabel("出生年月");
        JLabel address=new JLabel("家庭地址");
        JLabel cj=new JLabel("车间");
        JLabel zw=new JLabel("职位");
        final JTextField codeinput=new JTextField(10);
        final JTextField nameinput=new JTextField(10);
        final JTextField sexyinput=new JTextField(10);
        final JTextField ageinput=new JTextField(10);
        final JTextField birthdayinput=new JTextField(10);
        final JTextField addressinput=new JTextField(10);
        final JTextField gradeinput=new JTextField(10);
        final JTextField majorinput=new JTextField(10);
        JLabel title=new JLabel("基本信息");
    }
}

```



```

        JButton additionbutton=new JButton("添加");
        GridBagConstraints constraints=new GridBagConstraints();
        constraints.fill=GridBagConstraints.NONE;
        constraints.weightx=4;
        constraints.weighty=6;
        // 使用网格组布局添加控件
        add(title,constraints,0,0,4,1);
        add(name,constraints,0,1,1,1);
        add(code,constraints,0,2,1,1);
        add(sexy,constraints,0,3,1,1);
        add(age,constraints,0,4,1,1);
        add(nameinput,constraints,1,1,1,1);
        add(codeinput,constraints,1,2,1,1);
        add(sexyinput,constraints,1,3,1,1);
        add(ageinput,constraints,1,4,1,1);
        add(birthday,constraints,2,1,1,1);
        add(address,constraints,2,2,1,1);
        add(cj,constraints,2,3,1,1);
        add(zw,constraints,2,4,1,1);
        add(birthdayinput,constraints,3,1,1,1);
        add(addressinput,constraints,3,2,1,1);
        add(gradeinput,constraints,3,3,1,1);
        add(majorinput,constraints,3,4,1,1);
    }
}
```

上面程序代码的运行结果如图 13.16 所示。

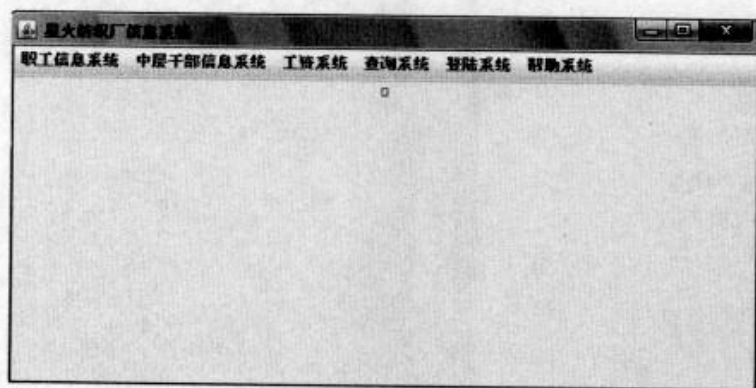


图 13.16 个性化菜单 (a)

当选择“职工信息系统”|“磨砂分厂职工信息”命令后，会弹出一个 JTabbedPane 容器，如图 13.17 所示。

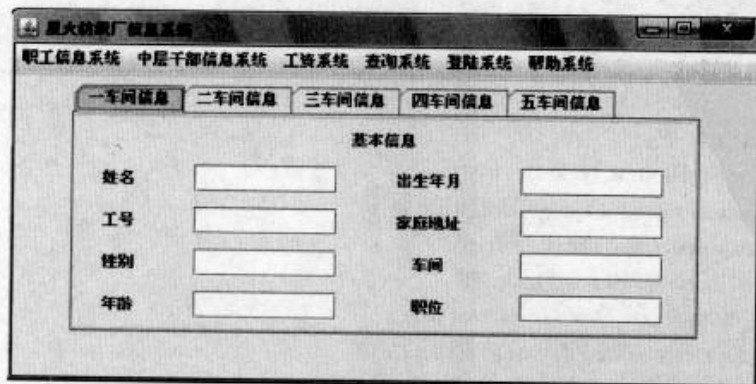


图 13.17 个性化菜单 (b)

上面的实例综合了菜单的创建、JTabbedPane 组件、布局管理等知识的应用，希望读者能够通过此实例将前面所学习到的知识进行融合，自己编写出一个具有个性化的菜单。当然在上面的实例中，有很多功能都没有实现，有兴趣的读者可以自己进行尝试。

13.1.10 菜单组件的常用 API

菜单组件中有很多 API 方法，如表 13.6 所示。

表 13.6 菜单组件常用 API

菜单组件	方法	说明
JMenu 组件	add(Component c)	将组件追加到此菜单的末尾
	add(Component c, int index)	将指定组件添加到此容器的给定位置上
	addMenuListener(MenuListener l)	添加菜单事件的侦听器
	addSeparator()	将新分隔符追加到菜单的末尾
	getItemCount()	返回菜单上的项数，包括分隔符
	getPopupMenu()	返回与此菜单关联的弹出菜单
	insert(JMenuItem mi, int pos)	在给定位置插入指定的 JMenuItem
	insertSeparator(int index)	在指定的位置插入分隔符
	menuSelectionChanged(boolean isIncluded)	当菜单栏选择更改为激活或取消激活此菜单时传递消息
	remove(Component c)	从此菜单移除组件 c
	remove(JMenuItem item)	从此菜单移除指定的菜单项
	setPopupMenuVisible(boolean b)	设置菜单弹出的可见性
	setSelected(boolean b)	设置菜单的选择状态
JMenuItem 组件	setEnabled(boolean b)	启用或禁用菜单项
	removeMenuDragMouseListener(MenuDragMouseListener l)	从菜单项中移除 MenuDragMouseListener
	setAccelerator(KeyStroke keyStroke)	设置组合键，它能直接调用菜单项的操作侦听器而不必显示菜单的层次结构
	addMenuKeyListener(MenuKeyListener l)	将 MenuKeyListener 添加到菜单项

当然，上面列出的表格只不过是一些常用方法的一部分，很多方法仍没有列出，但是并不代表不重要，有兴趣的读者可以参考一些 API 方面的书籍，并且配合 API 方法多多练习。

13.2 如何使用工具条组件

很多软件的主窗口上都会有一排工具，它的好处就是可以把用户经常使用的工具放到其中，从而方便用户快捷的操作，不用在众多菜单中搜索自己所需要的工具。本节将主要为读者讲述如何创建工具条。

13.2.1 如何创建工具条

本小节的主要目的是讲解如何创建工具条。Swing 中使用 JToolBar 类来创建工具条，JToolBar 的继承层次如图 13.18 所示。

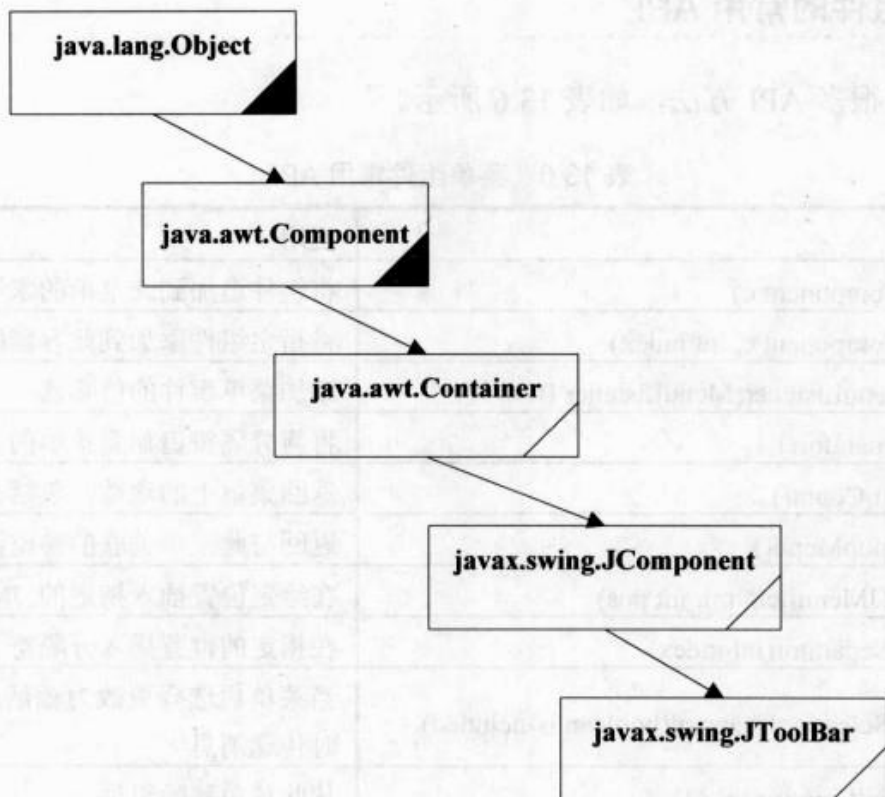


图 13.18 JToolBar 类层次图

从上面的层次图可知，JToolBar 只是一个普通的基本组件。JToolBar 用于放置各种常用的工具或控制组件，在设计软件时会将所有功能依类别放置在菜单中，但当功能数量相当多时，可能会造成用户进行一个简单的操作就必须繁琐地寻找菜单中相关的功能，从而造成用户操作上的负担。如果能将一般常用的功能以工具栏方式呈现在菜单下，不仅满足用户使用软件的意愿，也提高了工作效率。JToolBar 的构造器的说明如表 13.7 所示。

表 13.7 JToolBar 构造器

JToolBar 构造器	说明
JToolBar()	建立一个新的 JToolBar，位置为默认的水平方向
JToolBar(int orientation)	建立一个指定位置的 JToolBar
JToolBar(String name)	建立一个指定名称的 JToolBar
JToolBar(String name,int orientation)	建立一个指定名称和位置的 JToolBar

下面将通过一个简单实例讲解工具条的创建方法。其实例代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建一个工具条组件
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test9 extends JFrame
{
    private static final long serialVersionUID = 1L;
```



```
static final int WIDTH=600;
static final int HEIGHT=300;
public test9()
{
    super("工具条测试窗口");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // 创建三个普通按钮组件
    JButton button1 = new JButton("图标一");
    JButton button2 = new JButton("图标二");
    JButton button3 = new JButton("图标三");
    // 创建一个工具条, 再将上面创建的三个普通按钮组件添加到工具条中
    JToolBar bar = new JToolBar();
    bar.add(button1);
    bar.add(button2);
    bar.add(button3);
    // 创建一个文本区组件
    JTextArea edit = new JTextArea(8,40);
    JScrollPane scroller = new JScrollPane(edit);
    JPanel pane = new JPanel();
    BorderLayout bord = new BorderLayout();
    pane.setLayout(bord);
    pane.add("North",bar);
    pane.add("Center",scroller);
    setContentPane(pane);
    pack();
    setVisible(true);
}
public static void main(String args[])
{
    new test9();
}
```

上面程序代码的运行结果如图 13.19 所示。

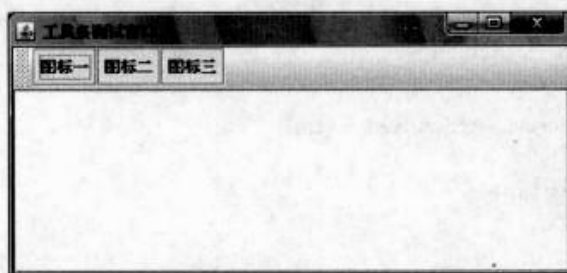


图 13.19 工具条的设计

工具条的设计非常简单,至于工具条的事件处理,则可以参考按钮控件的事件处理,因为工具条组件也是一个包含了多个按钮的容器。限于篇幅问题,这里不再详细讲述了,有兴趣的读者可以自行举例尝试。

13.2.2 如何定义个性化工具条

如何能够设计出个性化的工具条呢?首先必须要有扎实的基础知识,接着要尽量朝着用户使用快捷和方便的方向出发。这样的工具条才是具有个性化和实际意义。下面将通过一个比

较复杂的实例来展示如何设计一个带菜单和工具条的软件用户界面。其程序代码如下：

```
/* 这段程序代码主要为读者展示如何创建一个菜单和工具条组件，并且将菜单中的几个菜单项提取出来
 * 作为工具条中的工具按钮
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test10
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JPopupMenu pop;
    JMenuItem item2;
    JFrame f;
    JMenuItem item1;
    JPanel p;
    JTabbedPane tp;
    public test10()
    {
        ...
        // 与上述内容中相同实例的代码相似，这里不再列出
        JButton button1 = new JButton("磨砂分厂职工信息");
        JButton button2 = new JButton("纺纱分厂职工信息");
        JButton button3 = new JButton("人事部职工信息");
        JToolBar bar = new JToolBar();
        bar.add(button1);
        bar.add(button2);
        bar.add(button3);
        p.add("North",bar);
        ...
        // 与上述内容中相同位置的代码相似，这里不再列出
        // 处理工具条中第一个工具按钮的动作事件，其功能与前面的程序中的菜单项功能一样
        button1.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent Event)
            {
                // 这里不再详细列出
            }
        });
        item4.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent Event)
            {
                int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
                    "退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
                if(i==0)
                {
                    f.dispose();
                }
            }
        });
        menu1.addMenuListener(new MenuListener())
    }
}
```



```
{
    public void menuSelected(MenuEvent event)
    {
        item2.setEnabled(!item1.isSelected());
    }
    public void menuDeselected(MenuEvent event){}
    public void menuCanceled(MenuEvent event){}
});
}
public static void main(String args[])
{
    new test10();
}
}
class info extends JPanel
{
    ...
    // 这里不再详细列出
}
```

上面程序的运行结果如图 13.20 所示。

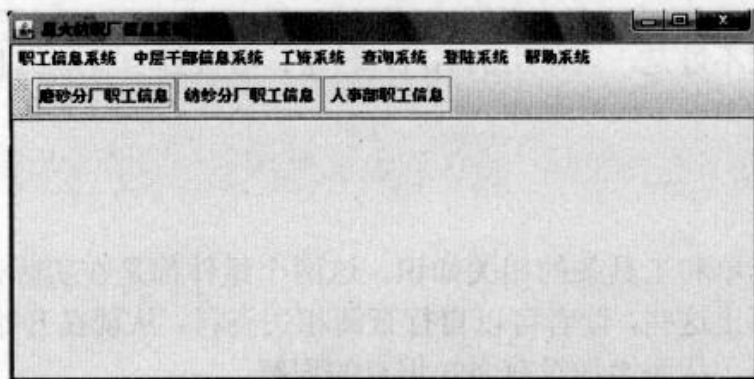


图 13.20 工具条的综合应用 (a)

当单击“磨砂分厂职工信息”按钮时，会弹出 JTabbedPane 容器，如图 13.21 所示。

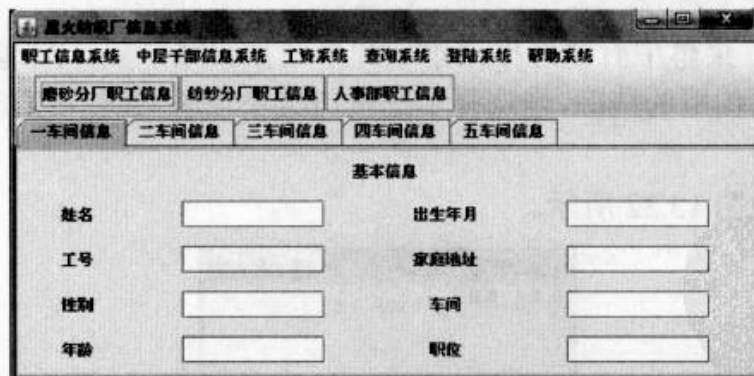


图 13.21 工具条的综合应用 (b)

上面的实例将工具条的事件转变成使用按钮事件，所以可以将工具条事件处理的新知识变成按钮控件事件的老知识。读者也可以将工具条中的按钮变成标签控件等，这就取决于用户对软件的要求。

13.2.3 工具条组件的常用 API

本小节将针对常用的工具条组件的 API 进行讲解，如表 13.8 所示。

表 13.8 工具条组件的常用 API

方法	说明
JToolBar()	创建一个工具栏，可选的 int 参数允许指定工具栏的方向，默认为 HORIZONTAL，可选的 String 参数，允许指定在非紧靠边框的工具栏窗口上显示的标题文本
JToolBar(int)	
JToolBar(String)	
JToolBar(String,int)	
Component add(Component)	在工具栏上添加一个组件
Void addSeparator()	在工具栏结尾添加一个分隔符
Void setFloatable(boolean)	在默认情况下，可浮动属性为 true，这表示用户可以将工具栏拖出并成为 一个单独的窗口
Boolean isFloatable	
Void setRollover(Boolean)	默认情况下 rollover 属性为 false，将其设置为 true 后将请求工具栏中的每个 按钮在鼠标处于其他位置时不具有边框
Boolean isRollover()	

上面所列的方法只不过是一些常用的 API，希望读者能够自行翻阅相关 API 方面的书籍。

13.3 本章小结

本章详细介绍了菜单和工具条的相关知识。这两个组件都是在实际开发中经常使用到的。当然它们的知识远远不止这些，读者可以自行查阅相关资料，从现在开始，读者就可以尝试着编写一些小的应用程序，从而增加对前面知识点的理解。

13.4 本章习题

1. 设计一个菜单，菜单中有复选框菜单项和单选按钮菜单项，当选择复选框菜单项时，会出现其他菜单项无法使用的情况，当选择单选按钮菜单项时，会使得所有的菜单项恢复使用。

要求：

(1) 其最终效果如图 13.22 所示。



图 13.22 本章习题 1 (a)

(2) 当选择“菜单一”|“让菜单项都不可用”命令时，所有的菜单项都会被禁止使用，如图 13.23 所示。

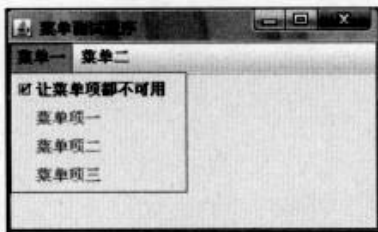


图 13.23 本章习题 1 (b)

(3) 当选择“菜单二”|“让菜单项都恢复可用”命令时，所有被禁止的菜单项都恢复了可用状态，并且“让菜单项都不可用”复选框被取消选中，如图 13.24 所示。

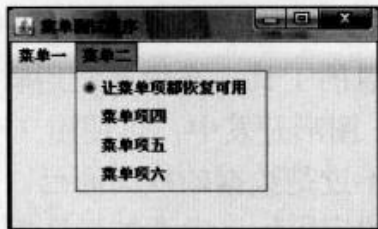


图 13.24 本章习题 1 (c)

2. 将上例中的菜单进行改进，每个菜单的第一项作为工具条中的工具项，并且要处理其中的动作事件。

要求：其最终效果如图 13.25 所示。

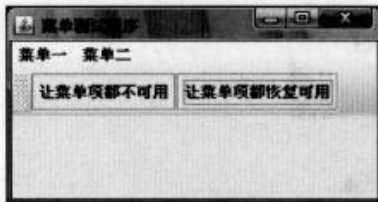


图 13.25 本章习题 2

此时工具条中的工具按钮与“菜单一”和“菜单二”中的第一项功能完全相同。

3. 将本章习题 1 中的菜单进行改进，每个菜单的第一项作为弹出式菜单的菜单项，并且处理其动作事件。

要求：其最终效果如图 13.26 所示。

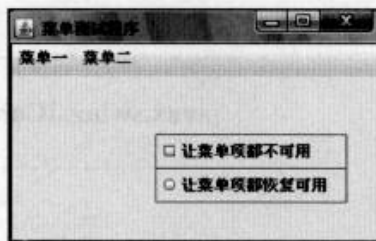


图 13.26 本章习题 3

4. 将本章习题 1 进行改进，使得菜单中每个菜单项都有快捷键。

要求：让“菜单项都不可用”的菜单项快捷键为 C，让“菜单项都恢复可用”的菜单项快捷键为 R，“菜单项一”的快捷键为 E，“菜单项二”的快捷键为 F，“菜单项三”的快捷键为 G，“菜单项四”的快捷键为 H，“菜单项五”的快捷键为 I，“菜单项六”的快捷键为 J。

第14章 如何使用表格组件

表格是用来描绘和存储数据信息的工具。在很多应用软件中，需要将数据以表格的形式展现在用户面前。在 Java 的 Swing 图形开发中，可使用 JTable 类来创建表格，但是 JTable 并没有包含或缓存任何数据，它只不过是数据的视图而已，同时，JTable 也允许用户去编辑表格。本章将会围绕如何在待开发的应用程序中添加表格展开详细讨论。

14.1 如何创建一个表格

本节将讲解如何创建表格，表格的层次继承如图 14.1 所示。

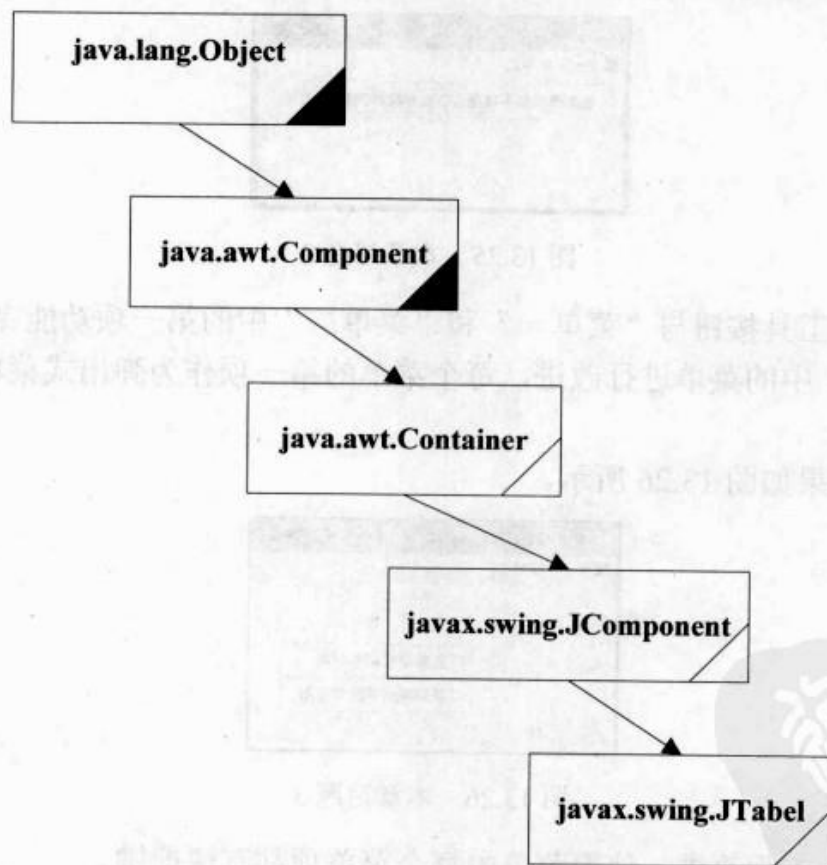


图 14.1 JTable 层次继承图

从上面的继承图可知，它不继承自 Window 类，所以它不可以独立显示，必须要依靠顶层容器类才能显示。表格组件的构造器说明如表 14.1 所示。

表 14.1 JTable 构造器

JTable 构造器	说明
JTable()	建立一个新的 JTable, 并使用系统默认的 Model
JTable(int numRows,int numColumns)	建立一个具有 numRows 行、numColumns 列的空表格, 使用的是 DefaultTableModel
JTable(Object[][] rowData, Object[] columnNames)	建立一个显示二维数组数据的表格且可以显示列的名称
JTable(TableModel dm)	建立一个 JTable, 有默认的字段模式以及选择模式, 并设置数据模式
JTable(TableModel dm,TableColumnModel cm)	建立一个 JTable, 设置数据模式与字段模式并有默认的选择模式
JTable(TableModel dm,TableColumnModel cm,ListSelectionModel sm)	建立一个 JTable, 设置数据模式、字段模式与选择模式
JTable(Vector rowData,Vector columnNames)	建立一个以 Vector 为输入来源的数据表格, 可显示行的名称

下面将以构造器 JTable(Object[][] rowData,Object[] columnNames)为例列举一个实例。这个实例主要是创建一个简单的表格, 其具体的实例代码如下所示:

```
// 这段程序代码主要是为读者展示创建表格的方法
// 创建表格的方法主要是通过 JTable(Object[][] rowData,Object[] columnNames)来创建两个数组作为创建表格的两个参数
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test1
{
    public test1()
    {
        JFrame f=new JFrame();
        Object[][] playerInfo=
        { // 创建表格中的数据
            {"王鹏",new Integer(91),new Integer(100),new Integer(191),new Boolean(true)},
            {"朱雪莲",new Integer(82),new Integer(69),new Integer(151),new Boolean(true)},
            {"梅庭",new Integer(47),new Integer(57),new Integer(104),new Boolean(false)},
            {"赵龙",new Integer(61),new Integer(57),new Integer(118),new Boolean(false)},
            {"李兵",new Integer(90),new Integer(87),new Integer(177),new Boolean(true)},
        };
        String[] Names={"姓名","语文","数学","总分","及格"}; // 创建表格中的横标题
        JTable table=new JTable(playerInfo,Names); // 以 Names 和 playerInfo 为参数, 创建一个表格
        table.setPreferredScrollableViewportSize(new Dimension(550,30));
        // 设置此表视口的首选大小
        JScrollPane scrollPane=new JScrollPane(table); // 将表格加入到滚动条组件中
        f.getContentPane().add(scrollPane,BorderLayout.CENTER);
        // 再将滚动条组件添加到中间容器中
        f.setTitle("表格测试窗口");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
        });
    }
}
```

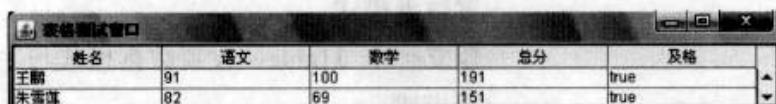


```

    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    test1 b=new test1();
}
}

```

上面程序代码的运行结果如图 14.2 所示。



姓名	语文	数学	总分	及格
王鹏	91	100	191	true
朱雪莲	82	69	151	true

图 14.2 表格的创建

上图中的表格是可以编辑的，所以可以创建一个新的空表格，然后在其中编辑好数据，此时就可以使用构造器 `JTable(int numRows,int numColumns)`来创建一个行数和列数确定的空表格。下面将列举一个实例，其具体代码如下所示：

```

// 这段程序代码主要是为读者展示创建表格的方法
// 通过 JTable(int numRows,int numColumns)来创建表格，numRows 指行数，numColumns 指列数。
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test2
{
    public test2()
    {
        JFrame f=new JFrame();
        JTable table=new JTable(10,10);// 创建一个 10 行和 10 列的空表格
        table.setPreferredScrollableViewportSize(new Dimension(550,30));
        JScrollPane scrollPane=new JScrollPane(table);
        f.getContentPane().add(scrollPane,BorderLayout.CENTER);
        f.setTitle("表格测试窗口");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        test2 b=new test2();
    }
}

```

上面程序代码的运行结果如图 14.3 所示。

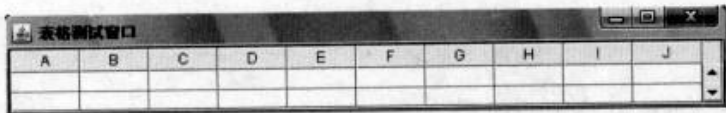


图 14.3 空表的创建

接下来，可以在空表格中进行编辑，如图 14.4 所示。

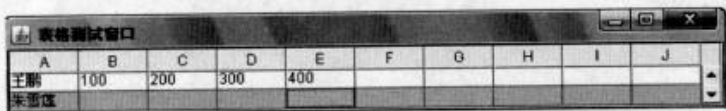


图 14.4 空表格的编辑

14.2 如何把表格加入容器

其实在上一节中，如何创建表格已经涉及到了将表格添加到容器中，将表格添加到容器的方法分成两种不同的情况，如表 14.2 所示。

表 14.2 把表格加入到容器的方法

把表格加入到容器的方法	说明
<code>JTable table=new JTable();</code> <code>JPanel pane=new JPanel();</code> <code>Pane.add(table);</code>	这种方法主要是将新创建好的表格直接添加到中间容器中。这种方法比较适合数据比较少的表格
<code>JTable table=new JTable();</code> <code>JScrollPane scroll=new JScrollPane(table);</code> <code>JPanel pane=new JPanel();</code> <code>Pane.add(scroll);</code>	这种方法是将新创建的表格先添加到滚动条组件中，然后将滚动条组件添加到中间容器中。这种方法比较适合数据比较多的表格，表格中数据一次性无法完全显示

下面将通过实例讲述如何将它添加到一般的容器中，其程序代码如下所示：

```
// 这段程序代码主要为读者展示如何将表格放到容器中，但是在这个表格中没有横标
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test3
{
    public test3()
    {
        JFrame f=new JFrame();
        Object[][] playerInfo=
        {
            {"王鹏",new Integer(91),new Integer(100),new Integer(191),new Boolean(true)},
            {"朱雪莲",new Integer(82),new Integer(69),new Integer(151),new Boolean(true)},
            {"梅庭",new Integer(47),new Integer(57),new Integer(104),new Boolean(false)},
            {"赵龙",new Integer(61),new Integer(57),new Integer(118),new Boolean(false)},
            {"李兵",new Integer(90),new Integer(87),new Integer(177),new Boolean(true)},
        };
        String[] Names={"姓名","语文","数学","总分","及格"};
        JTable table=new JTable(playerInfo,Names);
        table.setPreferredScrollableViewportSize(new Dimension(550,30));
        f.getContentPane().add(table,BorderLayout.CENTER); // 将表格直接添加到中间容器中
        f.setTitle("表格测试窗口");
    }
}
```

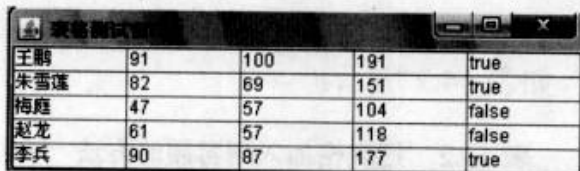


```

        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        test3 b=new test3();
    }
}

```

上面程序代码的运行结果如图 14.5 所示。



王鹏	91	100	191	true
朱雪莲	82	69	151	true
梅庭	47	57	104	false
赵龙	61	57	118	false
李兵	90	87	177	true

图 14.5 如何将表格添加到容器

不知道细心的读者是否发现，这个表格的横标题不见了，那么如何解决这个问题呢？除了使用上节中的 JScrollPane 组件来实现外，还可以使用其他的方法实现。实现代码如下所示：

```

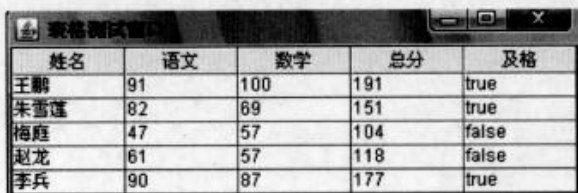
// 这段程序代码为读者展示如何将表格添加到容器中，并且添加横标
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test4
{
    public test4()
    {
        JFrame f=new JFrame();
        Object[][] playerInfo=
        {
            {"王鹏",new Integer(91),new Integer(100),new Integer(191),new Boolean(true)},
            {"朱雪莲",new Integer(82),new Integer(69),new Integer(151),new Boolean(true)},
            {"梅庭",new Integer(47),new Integer(57),new Integer(104),new Boolean(false)},
            {"赵龙",new Integer(61),new Integer(57),new Integer(118),new Boolean(false)},
            {"李兵",new Integer(90),new Integer(87),new Integer(177),new Boolean(true)},
        };
        String[] Names={"姓名","语文","数学","总分","及格"};
        JTable table=new JTable(playerInfo,Names);
        table.setPreferredScrollableViewportSize(new Dimension(550,30));
        f.getContentPane().add(table,BorderLayout.CENTER);
        f.getContentPane().add(table.getTableHeader(),BorderLayout.NORTH);
        f.setTitle("表格测试窗口");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {

```

// 添加表格的横标题


```
        System.exit(0);
    }
    });
}
public static void main(String[] args)
{
    test4 b=new test4();
}
}
```

上面程序代码的运行结果如图 14.6 所示。



姓名	语文	数学	总分	及格
王鹏	91	100	191	true
朱雪莲	82	89	151	true
梅庭	47	57	104	false
赵龙	61	57	118	false
李兵	90	87	177	true

图 14.6 带横标题的表格

由以上实例可知，可以使用如下代码在表格中添加一个横标题：

```
f.getContentPane().add(table.getTableHeader(),BorderLayout.NORTH);
```

现在来回忆一下两种创建表格的方式，一种是使用 JScrollPane 面板来添加表格，而另一种是使用中间容器直接添加表格，不过此时需要将表格的横标题放置在顶层窗口的最上层。为了能够更加熟悉将表格放入容器内的方法，下面再列举一个稍微复杂的实例。这个程序实例主要是创建一个菜单，通过对菜单的选择来打开表格，其具体的程序代码如下所示：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class test5
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JFrame f;
    JMenuItem item1;
    static JPanel p;
    JTable table;
    public test5()
    {
        ...
        // 与如何定义个性化菜单一节中实例中的创建菜单步骤内容相同，这里不再列出
        Object[][] playerInfo=
        {
            {"王鹏",new Integer(91),new Integer(1949),new Integer(1910)},
            {"朱雪莲",new Integer(82),new Integer(1969),new Integer(1510)},
            {"梅庭",new Integer(47),new Integer(1957),new Integer(1040)},
            {"赵龙",new Integer(61),new Integer(1957),new Integer(1180)},
            {"李兵",new Integer(90),new Integer(1987),new Integer(1770)},
        };
        String[] Names={"姓名","工号","出生年月","薪水"};
        table=new JTable(playerInfo,Names);
        table.setPreferredScrollableViewportSize(new Dimension(850,300));
```



```
item1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        table.setPreferredSize(new Dimension(850,300));
        p.add(table.getTableHeader(),BorderLayout.NORTH);
        p.add(table,BorderLayout.CENTER);
    }
});
}
public static void main(String args[])
{
    new test5();
}
```

上面程序代码的运行结果如图 14.7 所示。

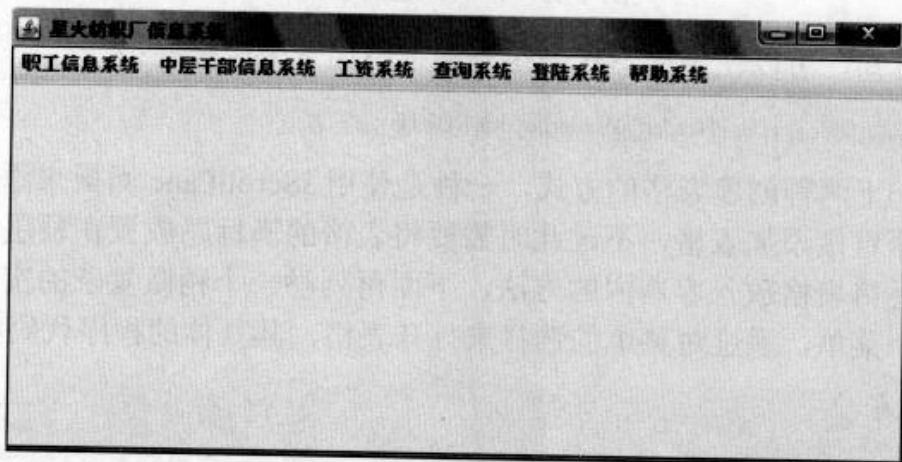


图 14.7 带菜单和表格的窗口 (a)

当选择“职工信息系统”|“磨砂分厂职工信息”命令时，会弹出如图 14.8 所示对话框。



图 14.8 带菜单和表格的窗口 (b)

从上面的实例可以看出，当要利用一个菜单项打开一个组件时，只需要将这个组件的构造器代码或者这个组件显示的代码放入菜单项的动作事件监听代码中即可。上面所有的程序代码中创建的表格还是比较粗糙的，例如无法自动调整表格的列宽等，所以在下一节将讨论如何调整表格的列宽。

14.3 如何设置表格列宽

前面所有的表格都无法自动调整列宽，只能在创建表格后，人工手动地调节每个列的宽度，那么如何才能做到自动调整列宽呢？从上面的运行结果可以发现，每个字段的宽度都是一样的，除非自行拉曳某个列宽。若想一开始就设置列宽的值，可以利用 `TableColumn` 类所提供的 `setPreferredWidth()` 方法来设置，并可利用 `JTable` 类所提供的 `setAutoResizeMode()` 方法来设置和调整某个列宽时其他列的列宽的变化情况。

在自动调整列的同时，其他列的可调整参数有 5 个，这 5 个参数的说明如表 14.3 所示。

表 14.3 自动调整列宽时其他列的变化参数

自动调整列宽时其他列的变化参数	说明
<code>AUTO_RESIZE_SUBSEQUENT_COLUMNS</code>	当调整某一列宽时，此字段之后的所有字段列宽都会跟着一起变动。此为系统默认值
<code>AUTO_RESIZE_ALL_COLUMNS</code>	当调整某一列宽时，此表格上所有字段的列宽都会跟着一起变动
<code>AUTO_RESIZE_OFF</code>	当调整某一列宽时，此表格上所有字段列宽都不会跟着改变
<code>AUTO_RESIZE_NEXT_COLUMN</code>	当调整某一列宽时，此字段的下一个字段的列宽会跟着改变，其余均不会变
<code>AUTO_RESIZE_LAST_COLUMN</code>	当调整某一列宽时，最后一个字段的列宽会跟着改变，其余均不会改变

下面将根据以上知识列举一个实例，该实例将添加表格自动调节列宽的功能，其实例代码如下所示：

```
// 这段程序代码主要是为读者展示如何创建一个能够自动调节列宽的表格
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.table.*;
public class test6
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JFrame f;
    JMenuItem item1;
    static JPanel p;
    JTable table;
    TableColumn column;
    public test6()
    {
        ...
        // 与上述内容中相同位置的代码相似，这里不再列出
        Object[][] playerInfo=
        {
            {"王鹏",new Integer(91),new Integer(1949),new Integer(1910)},
            {"朱雪莲",new Integer(82),new Integer(1969),new Integer(1510)},
            {"梅庭",new Integer(47),new Integer(1957),new Integer(1040)},
            {"赵龙",new Integer(61),new Integer(1957),new Integer(1180)},
        }
    }
}
```



```

        {"李兵",new Integer(90),new Integer(1987),new Integer(1770)},
    };
    String[] Names={"姓名","工号","出生年月","薪水"};
    table=new JTable(playerInfo,Names);
    table.setPreferredScrollableViewportSize(new Dimension(850,300));
    p.add(table.getTableHeader(),BorderLayout.NORTH);
    p.add(table,BorderLayout.CENTER);
    table.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
    // 让其他的列也跟着变动
    for (int i=0;i<5;i++)
    {
        // 下面的 for 语句用于规范表格中的偶数列宽和奇数列宽
        column=table.getColumnModel().getColumn(i);
        if ((i%2)==0)
            column.setPreferredWidth(100);
        else
            column.setPreferredWidth(50);
    }
}

public static void main(String args[])
{
    new test6();
}
    
```

上面程序的运行结果如图 14.9 所示。



图 14.9 如何自动调节列宽

在上面的程序中，当列数为奇数时，会自动调整其列宽为 50，当列数为偶数时，会自动调整其列宽为 100，并且使用的其他列参数是当指定列变化时，其后面的列都紧随着其变化。有兴趣的读者可以尝试设置其他 4 个参数，这里不再赘述。

14.4 如何创建表格模型

什么是表格模型呢？这就是本节需要讲解的内容。TableModel 是一个接口，在这个接口里定义了若干的方法，其中包括存取表格字段（cell）的内容、计算表格的列数等基本存取操作，从而可以让设计者简单地利用 TableModel 来创建一个自己想要的表格。

TableModel 界面存放在 javax.swing.table 包中，这个包中定义了许多 JTable 会用到的各种 Model。常用的方法如表 14.4 所示。

表 14.4 表格模型的常用方法

方法	说明
void addTableModelListener (TableModelListener l);	使表格具有处理 TableModelEvent 的能力。当表格的 TableModel 有所变化时，会发出 TableModel Event 事件信息
Class getColumnClass(int columnIndex);	返回字段数据类型的类名称
int getColumnCount();	返回字段（行）数量
String getColumnName(int columnIndex);	返回字段名称
int getRowCount();	返回数据列数量
Object getValueAt(int rowIndex,int columnIndex);	返回数据某个 cell 中的值
Boolean isCellEditable(int rowIndex,int columnIndex);	返回 cell 是否可编辑，为 true 则可编辑
void removeTableModelListener(TableModelListener l);	从 TableModelListener 中移除一个 listener
Void setValueAt(Object aValue,int rowIndex,int columnIndex);	设置某个 cell(rowIndex,columnIndex)的值

由于 TableModel 本身是一个接口，因此，若要直接实现此接口来建立表格并不容易。幸好 Java 提供了两个类分别实现了这个界面，一个是 AbstractTableModel 抽象类，一个是 DefaultTableModel 实体类。前者实现了大部分的 TableModel 方法，让用户可以自由地构造自己的表格模式。而后者继承前者的类，是默认的表格模式。

AbstractTableModel 是一个抽象类，这个类已经实现了大部分的 TableModel 方法（除了 getRowCount()、getColumnCount()、getValueAt()这三个方法），所以要利用 AbstractTableModel 类创建表格，关键就是要去实现这三个方法，才能利用此抽象类设计出不同格式的表格。

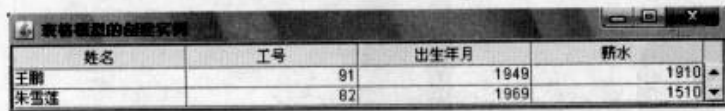
下面将通过实例讲述如何通过使用 AbstractTableModel 类来创建一个表格，其具体的实例代码如下：

```
// 这段程序代码主要是为读者展示如何使用 AbstractTableModel 类来创建一个表格
import javax.swing.table.AbstractTableModel;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test7
{
    public test7()
    {
        JFrame f = new JFrame();
        MyTable mt=new MyTable();
        JTable t=new JTable(mt);
        t.setPreferredScrollableViewportSize(new Dimension(550, 30));
        JScrollPane s = new JScrollPane(t);
        f.getContentPane().add(s, BorderLayout.CENTER);
        f.setTitle("表格模型的创建实例");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
}
```



```
}
public static void main(String args[])
{
    new test7();
}
}
class MyTable extends AbstractTableModel
{
    // 创建一个类 MyTable, 它继承自 AbstractTableModel 类
    // 在这个类中实现 getRowCount()、getColumnCount()、getValueAt()这三个方法
    Object[][] p=
    {
        {"王鹏",new Integer(91),new Integer(1949),new Integer(1910)},
        {"朱雪莲",new Integer(82),new Integer(1969),new Integer(1510)},
        {"梅庭",new Integer(47),new Integer(1957),new Integer(1040)},
        {"赵龙",new Integer(61),new Integer(1957),new Integer(1180)},
        {"李兵",new Integer(90),new Integer(1987),new Integer(1770)},
    };
    String[] n = {"姓名","工号","出生年月","薪水"};
    public int getColumnCount()                // 此方法是返回该模型中的列数
    {
        return n.length;
    }
    public int getRowCount()                    // 此方法是返回该模型中的行数
    {
        return p.length;
    }
    public String getColumnName(int col)        // 此方法是返回 col 位置的列的名称
    {
        return n[col];
    }
    public Object getValueAt(int row, int col) // 此方法是返回 row 和 col 的单元格的值
    {
        return p[row][col];
    }
    public Class getColumnClass(int c)          // 此方法是针对列中所有的单元格值, 返回最具体的超类
    {
        return getValueAt(0, c).getClass();
    }
}
```

上面程序的运行结果如图 14.10 所示。



姓名	工号	出生年月	薪水
王鹏	91	1949	1910
朱雪莲	82	1969	1510

图 14.10 利用表格模型创建表格

上面所讲述的利用 AbstractTableModel 类来创建表格,其难度在于要实现这个抽象类中的一些方法,为了能够让开发人员更加简便地创建表格,下面将讲述 DefaultTableModel 的用法。

DefaultTableModel 比 AbstractTableModel 要简单许多,也常常在实际开发中使用。DefaultTableModel 内部使用 Vector 数据结构来存储表格的数据,若要显示的表格格式比较简单,可以使用 DefaultTableModel 类。若要显示的数据模式非常复杂,例如表格显示的信息因人而异,这种情况可以使用 AbstractTableModel 类。

DefaultTableModel 类的构造器的说明如表 14.5 所示。

表 14.5 DefaultTableModel 类的构造器

DefaultTableModel 类的构造器	说明
DefaultTableModel()	建立一个 DefaultTableModel, 里面没有任何数据
DefaultTableModel(int numRows,int numColumns)	建立一个指定行、列数的 DefaultTableModel
DefaultTableModel(Object[][] data, Object[] columnNames)	建立一个 DefaultTableModel, 输入数据格式为 Object Array。系统会自动调用 setDataVector()方法来设置数据
DefaultTableModel(Object[] columnNames,int numRows)	建立一个 DefaultTableModel, 并具有 Column Header 名称与行数信息
DefaultTableModel(Vector columnNames,int numRows)	建立一个 DefaultTableModel, 并具有 column Header 名称与行数信息
DefaultTableModel(Vector data, Vector columnNames)	建立一个 DefaultTableModel, 输入数据格式为 Vector。系统会自动调用 setDataVector()方法来设置数据

下面将列举一个实例来讲解如何使用 DefaultTableModel 类来创建表格, 其程序代码如下所示:

```
// 这段程序代码主要是为读者展示如何使用 DefaultTableModel 类创建表格
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

public class test8 implements ActionListener
{
    JTable table = null;
    DefaultTableModel defaultModel = null;
    public test8()
    {
        JFrame f = new JFrame();
        Object[][] p =
        {
            {"王鹏",new Integer(91),new Integer(1949),new Integer(1910)},
            {"朱雪莲",new Integer(82),new Integer(1969),new Integer(1510)},
            {"梅庭",new Integer(47),new Integer(1957),new Integer(1040)},
            {"赵龙",new Integer(61),new Integer(1957),new Integer(1180)},
            {"李兵",new Integer(90),new Integer(1987),new Integer(1770)},
        };
        String[] n = {"姓名","工号","出生年月","薪水"};
        defaultModel = new DefaultTableModel(p,n);           // 创建一个默认的表格模型
        table=new JTable(defaultModel);
        table.setPreferredScrollableViewportSize(new Dimension(400, 80));
        JScrollPane s = new JScrollPane(table);
        JPanel panel = new JPanel();
        JButton b = new JButton("增加行");
        panel.add(b);
        panel.add(b);
        b.addActionListener(this);
        b = new JButton("删除行");
        panel.add(b);
    }
}
```



```
panel.add(b);
b.addActionListener(this);
Container contentPane = f.getContentPane();
contentPane.add(panel, BorderLayout.NORTH);
contentPane.add(s, BorderLayout.CENTER);
f.setTitle("AddRemoveCells");
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public void actionPerformed(ActionEvent e)
{
    if(e.getActionCommand().equals("增加行"))
        // 如果单击“增加行”按钮后, 会在表中增加一行
        defaultModel.addRow(new Vector());
    if(e.getActionCommand().equals("删除行"))
        // 如果单击“删除行”按钮后, 会在表中删除所选中的一行, 并且设置下一行为当前行
        {
            int rowcount = defaultModel.getRowCount()-1;
            // getRowCount 返回行数, rowcount<0 代表已经没有任何行了
            if(rowcount >= 0)
            {
                defaultModel.removeRow(rowcount);
                defaultModel.setRowCount(rowcount);
                /* 删除行比较简单, 只要用 DefaultTableModel 的 removeRow()方法即可
                * 删除行完毕后必须重新设置列数, 也就是使用 DefaultTableModel 的 setRowCount()方法来设置当前行
                */
            }
        }
    table.revalidate();
}
public static void main(String args[])
{
    new test8();
}
```

上面程序代码的运行结果如图 14.11 所示。



姓名	工号	出生年月	薪水
王鹏	91	1949	1910
朱雪莲	82	1969	1510
梅庭	47	1957	1040
赵龙	61	1957	1180
李兵	90	1987	1770

图 14.11 使用 DefaultTableModel 类创建表格

当单击“增加行”按钮时, 会出现如图 14.12 所示界面。



图 14.12 单击“增加行”按钮后的界面

以上程序只不过是一个按钮的事件引起表格的变化。在 Excel 中，当一个表格中的一个数据发生变化时，则整个表格其他相关的数据都会发生变化，那么，这种功能是如何实现的呢？下一节将进行讲解。

14.5 如何监听数据变化

在 Excel 表中，当表格中的一个单元格的数据发生变化时，与之相关的数据也会同时发生变化，这就涉及到了表格监听器的知识。JTable 的事件大致均针对表格内容的操作处理，包括字段内容改变、列数增加或减少、行数增加或减少、表格的结构改变等。这些事件称为 TableModelEvent 事件。要处理 TableModelEvent 事件必须实现 TableModelListener，此接口定义了一个方法，即 tableChanged()。

下面将给出一个实例，该实例主要用于创建一个表格，通过对这个表格中的某个单元格数据进行修改，使得其他与之相关的单元格数据发生变化。其实例代码如下所示：

```
// 这段程序代码主要是为读者展示如何处理 TableModelEvent 事件
import javax.swing.table.AbstractTableModel;
import javax.swing.event.*;
import javax.swing.table.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class test9 implements TableModelListener
{
    JTable table = null;
    MyTable my = null;
    public test9()
    {
        JFrame f = new JFrame();
        my = new MyTable();
        my.addTableModelListener(this);
        table = new JTable(my);
        table.setPreferredSize(new Dimension(550, 30));
        JScrollPane s = new JScrollPane(table);
        f.getContentPane().add(s, BorderLayout.CENTER);
        f.setTitle("表格事件处理");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
}
```



```

    }
    public void tableChanged(TableModelEvent e)
    {
        // 处理表格数据变化事件
        int row = e.getFirstRow();
        int grade1=((Integer)(my.getValueAt(row,1))).intValue();
        int grade2=((Integer)(my.getValueAt(row,2))).intValue();
        int total = grade1+grade2;// 将两次获得的值进行叠加
        my.mySetValueAt(new Integer(total),row,3);
        table.repaint();// 系统重新绘制表格
    }
    public static void main(String args[])
    {
        new test9();
    }
}
class MyTable extends AbstractTableModel
{
    // 使用 AbstractTableModel 来创建表格模型
    Object[][] p =
    {
        {"王鹏",new Integer(91),new Integer(100),new Integer(191)},
        {"朱雪莲",new Integer(82),new Integer(100),new Integer(182)},
        {"梅庭",new Integer(47),new Integer(100),new Integer(147)},
        {"赵龙",new Integer(61),new Integer(100),new Integer(161)},
        {"李兵",new Integer(90),new Integer(100),new Integer(190)},
    };
    String[] n = {"姓名","语文","数学","总分"};
    public int getColumnCount()
    {
        return n.length;
    }
    public int getRowCount()
    {
        return p.length;
    }
    public String getColumnName(int col)
    {
        return n[col];
    }
    public Object getValueAt(int row, int col)
    {
        return p[row][col];
    }
    public Class getColumnClass(int c)
    {
        return getValueAt(0, c).getClass();
    }
    public boolean isCellEditable(int rowIndex, int columnIndex)
    {
        // 判断单元格是否可以编辑
        return true;
    }
    public void setValueAt(Object value, int row, int col)
    {
        p[row][col] = value;
        fireTableCellUpdated(row, col);
    }
    public void mySetValueAt(Object value, int row, int col)

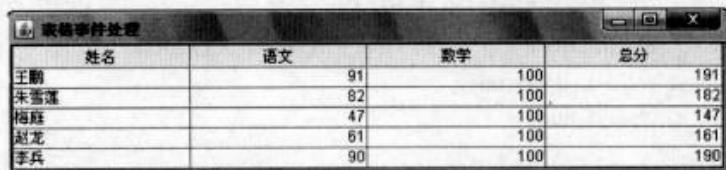
```

// 获得所选数据的行数
// 获得此行第 2 列的值
// 获得此行第 3 列的值

// 将变化的值赋给第 4 列


```
{  
    p[row][col] = value;  
}
```

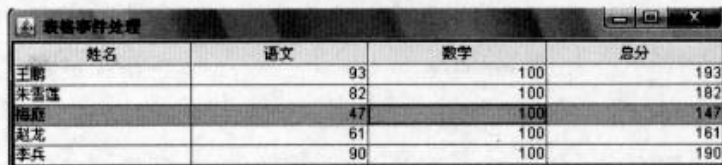
上面程序的运行结果如图 14.13 所示。



姓名	语文	数学	总分
王鹏	91	100	191
朱雪莲	82	100	182
梅庭	47	100	147
赵龙	61	100	161
李兵	90	100	190

图 14.13 表格事件处理 (a)

当将语文成绩改变后，总成绩也会跟着变化，如图 14.14 所示。



姓名	语文	数学	总分
王鹏	93	100	193
朱雪莲	82	100	182
梅庭	47	100	147
赵龙	61	100	161
李兵	90	100	190

图 14.14 表格事件处理 (b)

上面的实例程序讲解了如何处理表格事件，其实还有很多相关的内容，限于篇幅，这里不再赘述，希望读者能够查阅相关资料，自行举例多多练习，从而真正掌握其用法。

14.6 如何使用选择器

选择器是指表格的选择模式 `SelectionMode`。选择器的最大用处就是使用户能够以不同的方式选择表中的数据，例如平时处理 Excel 表时，可以一次性选择一个数据，也可以一次性选择多个数据等。选择器的操作方式与 `JList` 操作方式极为相似，包括其事件驱动。为了能够让读者熟悉选择器的用法，在这里将给出一个实例。该实例通过编写代码来实现对表格中数据的不同选择方式，如一次选择一个数据、一次选择相连的几个数据、一次选择不相连的多个数据等，其具体的实例代码如下所示：

```
// 这段程序代码主要是展示如何创建选择器，和使用选择器来以不同的方式选择表格中的单元格  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
import javax.swing.event.*;  
public class test10 implements ActionListener, ListSelectionListener  
{  
    JTable table=null;  
    ListSelectionModel selectionMode=null;  
    JLabel label=null;// 显示用户选取表格之用  
    public test10()  
    {  
        JFrame f=new JFrame();  
        Object[][] p =  
        { { "王鹏","91","100","191"},  
          { "朱雪莲","82","100","182"},  
          { "梅庭","47","100","147"},  
          { "赵龙","61","100","161"},  
        }  
    }  
}
```



```

        {"李兵","90","100","190"},
    };
    String[] n = {"姓名","语文","数学","总分"};
    table=new JTable(p,n);
    table.setPreferredScrollableViewportSize(new Dimension(400,80));
    table.setCellSelectionEnabled(true);// 使得表格的选取是以 cell 为单位而不是以列为单位
    // 若没有写此行,则在选取表格数据时以整列为单位
    selectionMode=table.getSelectionModel();// 取得 table 的 ListSelectionModel.
    selectionMode.addListSelectionListener(this);
    JScrollPane s=new JScrollPane(table);
    JPanel panel=new JPanel();
    JButton b=new JButton("单一选择");
    panel.add(b);
    b.addActionListener(this);
    b=new JButton("连续区间选择");
    panel.add(b);
    b.addActionListener(this);
    b=new JButton("多重选择");
    panel.add(b);
    b.addActionListener(this);
    label=new JLabel("你选取的数据是:");
    Container contentPane=f.getContentPane();
    contentPane.add(panel,BorderLayout.NORTH);
    contentPane.add(s,BorderLayout.CENTER);
    contentPane.add(label,BorderLayout.SOUTH);
    f.setTitle("选择器的测试实例");
    f.pack();
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    });
}

public void actionPerformed(ActionEvent e)
{
    /*处理按钮事件,利用 ListSelectionModel 界面所定义的 setSelectionMode()方法来设置表格选取模式.*/
    if (e.getActionCommand().equals("单一选择"))
        selectionMode.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    if (e.getActionCommand().equals("连续区间选择"))
        selectionMode.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
    if (e.getActionCommand().equals("多重选择"))
        selectionMode.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
    table.revalidate();
}

public void valueChanged(ListSelectionEvent e)
/* 当用户选取表格数据时会触发 ListSelectionEvent,使用 ListSelectionListener 界面来处理这一事件
* ListSelectionListener 界面只定义一个方法,那就是 valueChanged()
*/
{
    String tempString="";
    // JTable 的 getSelectedRows()与 getSelectedColumns()方法会返回
    // 已选取表格 cell 的 index Array 数据
    int[] rows=table.getSelectedRows();
    int[] columns=table.getSelectedColumns();

```



```
for (int i=0;i<rows.length;i++)
{
    // JTable 的 getValueAt()方法会返回某行的 cell 数据，返回值是 Object 数据类型
    // 因此要自行转成 String 数据类型
    for (int j=0;j<columns.length;j++)
        tempString = tempString+" "+(String)table.getValueAt(rows[i], columns[j]);
}
label.setText("你选取的数据是:"+tempString);
}

public static void main(String[] args)
{
    new test10();
}
```

上面程序代码的运行结果如图 14.15 所示。



图 14.15 选择器的使用 (a)

当单击“单一选择”按钮时，按住 Ctrl 或者 Shift 键，一次性只能选择一个单元格数据，如图 14.16 所示。



图 14.16 选择器的使用 (b)

当单击“连续区间选择”按钮时，按住 Ctrl 或者 Shift 键，一次性可以选择连续区间的数据，如图 14.17 所示。



图 14.17 选择器的使用 (c)

当单击“多重选择”按钮时，按住 Ctrl 或者 Shift 键，一次性可以选择所有的数据，如图 14.18 所示。



图 14.18 选择器的使用 (d)

以上实例对选择器的三种情况进行了详细介绍, 其中有三种模式, 其意义如表 14.6 所示。

表 14.6 选择器的三种模式

选择器的三种模式	说明
static int SINGLE_SELECTION	单个单元格的选择
static int SINGLE_INTERVAL_SELECTION	连续区间的单元格的选择
Static int MULTIPLE_INTERVAL_SELECTION	多重单元格的选择, 无论是连续区间或者不连续区间

上表中的三种模式可分别让用户进行单一选择、连续区间选择、多重选择。当用户进行后两个模式的操作时, 应配合 Shift 键或 Ctrl 键。

14.7 如何使用编辑器和渲染器

编辑器主要是使表格中的数据处于可编辑状态。在默认状态下, 每个表格的单元格都可以使用下拉列表框等组件编辑数据。在实际开发中, 可以在每个单元格中放入一些可以编辑的组件。

渲染器用来绘制表格中的单元格, 渲染器也可以从表格中提取每个单元格的方法, 所以, 渲染器也被称作绘制器。其实, 当一个表格能够在容器中显示出来, 是依靠系统默认的渲染器来绘制的, 而且是一个单元格、一个单元格的绘制, 所以, 它可以单独的将某个单元格提取出来, 也可以针对表格中任意的单元进行编辑, 这也是渲染器的最大用处之一。

上述的理论, 读者不一定能够清楚, 下面将给出一个有关渲染器的实例, 使读者能够从实例中充分了解渲染器的使用方法和步骤。这个实例主要是通过渲染器来渲染表格中不同行的颜色。其程序代码如下所示:

```
// 这段程序代码主要是为读者展示使用渲染器让表格中偶数行的颜色为蓝色, 奇数行不变
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;

public class test11 extends JFrame
{
    Object[][] p =
    {
        {"王鹏", "91", "100", "191"},
        {"朱雪莲", "82", "100", "182"},
        {"梅庭", "47", "100", "147"},
        {"赵龙", "61", "100", "161"},
        {"李兵", "90", "100", "190"},
    };

    String[] n = {"姓名", "语文", "数学", "总分"};
```



```

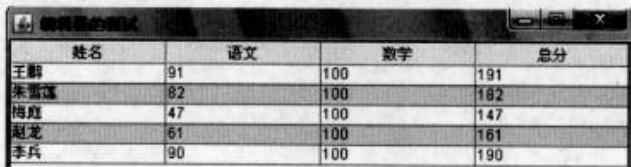
// 界面组件-----
private JScrollPane scroPanel = new JScrollPane(); // 中底层滚动面板
private DefaultTableModel model; // 列表默认 TableModel
private JTable table;
public test()
{
    config();
    addListener();
    confcolor();
}
/**
 * 方法: 界面构建
 */
private void config()
{
    table = new JTable(model = new DefaultTableModel(p,n));
    DefaultTableCellRenderer tcr = new DefaultTableCellRenderer()
    {
        public Component getTableCellRendererComponent(JTable table, Object value,
            boolean isSelected, boolean hasFocus, int row, int column)
        {
            // 设置偶数行颜色
            if(row%2 == 0)
                setBackground(Color.white);
            // 设置奇数行颜色
            else if(row%2 == 1)
                setBackground(new Color(206,231,255));
            return super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);
        }
    };
    for(int i=0;i<=3;i++)
    {
        table.getColumnModel().getColumn(i).setCellRenderer(tcr);
    }
    scroPanel.getViewport().setBackground(Color.white);
    scroPanel.getViewport().add(table);
    // 总体界面布局-----
    getContentPane().add(scroPanel, BorderLayout.CENTER);
}
/**
 * 方法: 界面显示
 */
private void confcolor()
{
    setTitle("编辑器的测试");
    setSize(500,400);
    Toolkit tmpTK = Toolkit.getDefaultToolkit();
    Dimension dime = tmpTK.getScreenSize();
    Dimension frameSize = this.getPreferredSize();
    setLocation(dime.width/2 - (frameSize.width/2),
        dime.height/2 - (frameSize.height/2));
    setResizable(false);
    setVisible(true);
}
/**
 * 方法: 添加事件监听 addListener()

```



```
*/
private void addListener()
{
    this.addWindowListener(new WindowAdapter()
    {
        // 添加窗口关闭事件
        public void windowClosing(WindowEvent e)
        {
            new JFrame().setVisible(false);
            dispose();
        }
    });
}
/**
 * 程序入口 main()
 */
public static void main(String args[])
{
    new test();
}
```

上面程序代码的运行结果如图 14.9 所示。



姓名	语文	数学	总分
王鹏	91	100	191
朱雪莲	82	100	182
梅庭	47	100	147
赵龙	61	100	161
李兵	90	100	190

图 14.9 间隔色的形成

JTable 的 API 并没有提供更改表格行或列的颜色的能力。表格的表头和内容呈现的形式都是由相应的 Renderer（渲染器）来控制的，所以，只需要继承单元格默认的 Renderer 并作相应的修改就可以达到目的，以上的程序使用的方法如下：

```
table = new JTable(model = new DefaultTableModel(p,n));
DefaultTableCellRenderer tcr = new DefaultTableCellRenderer()
{
    public Component getTableCellRendererComponent(JTable table, Object value,
        boolean isSelected, boolean hasFocus, int row, int column)
    {
        if(row%2 == 0)
            setBackground(Color.white);
        else if(row%2 == 1)
            setBackground(new Color(206,231,255));
        return super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);
    }
};
```

若要实现接口 TableCellRenderer，则必须要实现这个接口中惟一的方法 getTableCellRendererComponent()。

针对上面的实例，只需要调用下面的函数，就可以保证表格在内容被更改之后依然正确显示间隔色：


```

/** 为所有表格设置间隔色 */
private void setRenderColor()
{
    for( int i = 0;i<table.getColumnModel().getColumnCount();i++)
        table.getColumnModel().getColumn( colname[i] ).setCellRenderer(colorRender );
}

```

以上的代码段说明了渲染器可以让表格中的数据具有不同的表现形式。整个程序实例突出表现了渲染器能够操纵表格中任意单元的特点，所以，在实际开发中，如果遇到要修改或者编辑表格中的单元格的时候，读者可以使用渲染器来编写代码。

下面将给出一个有关编辑器的实例，这个实例主要是让表格中的第一行不可编辑，其具体程序代码如下所示：

// 这段程序代码主要是为读者展示如何使用编辑器让第一列数据为不可编辑状态

```

import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
public class test12
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("编辑器测试");
        frame.setSize(300,200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new BorderLayout());
        Object[][] p =
        {
            {"王鹏","91","100","191"},
            {"朱雪莲","82","100","182"},
            {"梅庭","47","100","147"},
            {"赵龙","61","100","161"},
            {"李兵","90","100","190"},
        };
        String[] n = {"姓名","语文","数学","总分"};
        JTable table = new JTable();
        frame.getContentPane().add(new JScrollPane(table));
        frame.setVisible(true);
        table.setModel(new DefaultTableModel(p,n)
        {
            public boolean isCellEditable(int rowIndex,int columnIndex)
            {
                if(columnIndex==0)
                    return false;
                return true;
            }
        });
    }
}

```

上面程序代码的运行结果如图 14.20 所示。

姓名	语文	数学	总分
王鹏	91	100	191
朱雪莲	82	100	182
梅庭	47	100	147
赵龙	61	100	161
李兵	90	100	190

图 14.20 不可编辑的单元格

上面的程序让第一列姓名不可编辑，实现单元格能否被编辑取决于 JTable 的 isCellEditable(int row,int column)。如果该函数返回 true，则(row,column)所代表的单元格可以被编辑，否则该单元格不能被编辑。在程序中，可使用如下代码来决定哪一行或者哪一列不可编辑：

```
public boolean isCellEditable(int rowIndex,int columnIndex)
{
    if(columnIndex==0)
        return false;
    return true;
}
```

在实际开发中，使用编辑器或者渲染器来设计表格的实例很多，在这里限于篇幅，不再一一举例，希望读者能够自行举例练习。

14.8 如何使用自定义渲染器

如何自定义渲染器呢？在这里首先将介绍表格渲染器的一些基本知识。TableCellRenderer 接口定义了 JTable 渲染器接口，其接口定义的具体内容如下所示：

```
public interface TableCellRenderer
{
    /* 返回渲染表格的组件，使用该方法在渲染之前配置渲染器 */
    Component getTableCellRendererComponent(JTable table,Object value,boolean isSelected,boolean hasFocus,
        int row,int column);
}
```

针对上面接口中的各个参数，下面将以表格的形式列出其意义，如表 14.7 所示。

表 14.7 TableCellRenderer 接口的参数表

参数	说明
table	请求渲染器渲染的表，可以为空
value	要渲染的单元格的值，由渲染器来决定如何解释和渲染该值，value 的值为字符串"true"，渲染器可以渲染成字符串，也可以渲染成一个 check box，值可以为空
isSelected	当前表格是否被选中，渲染器应据此决定是否高亮显示
hasFocus	当前表格是否拥有焦点，渲染器应据此进行特殊渲染，如画一个虚线框
row	当前表格的行号。如果渲染的是表头，该值为-1
column	当前表格的列号

如何使用渲染器接口来实现对组件的扩展呢？JTable 中许多方法可用来进行渲染器的设置，也就是说将渲染器对象作为一个方法的参数，来实现针对表格中单元格的不同设置，如下方法：

```
public void setDefaultRenderer(Class columnClass, TableCellRenderer renderer);
```

下面是一个很短的代码，它实现了在表的第 2 行、第 3 行中插入一个表框。其实例代码如下所示：

```
// 这段程序代码主要展示了如何利用渲染器在表中再插入一个表格
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;

public class test13 extends JTable
{
    public test13()
    {
        JFrame f=new JFrame("测试窗口");
        f.setVisible(true);
        f.pack();
        Object[][] p =
        {
            {"王鹏","91","100","191"},
            {"朱雪莲","82","100","182"},
            {"梅庭","47","100","147"},
            {"赵龙","61","100","161"},
            {"李兵","90","100","190"},
        };
        String[] n = {"姓名","语文","数学","总分"};
        // 界面组件
        JScrollPane scroPanel = new JScrollPane(this); // 中底层滚动面板
        f.getContentPane().add(scroPanel, BorderLayout.CENTER);
        setModel(new DefaultTableModel(p,n));
        setRowHeight(1,super.getRowHeight()*4);
        // 将第 2 行高度设置宽一些，使嵌入的表格显示起来好看些
    }

    public TableCellRenderer getCellRenderer(int row, int column)
    { // 重载 getCellRenderer 提供自己的 TableCellRenderer
        if(row == 1 && column==2)
        {
            return new TableCellRenderer()
            { // 在第 2 行、第 3 列提供一个子表的渲染器
                JTable subTable=new JTable(new DefaultTableModel(4,4));
                // 子表可以自己定制内容
                public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
                    boolean hasFocus, int row, int column)
                { // 实现 TableCellRenderer 的方法，提供该子表渲染器
                    return subTable;
                }
            };
        }
        else
            return super.getCellRenderer(row, column);
        // 如果是其他地方的表格，沿用父类中提供的渲染器
    }
}
```



```
...  
}  
public static void main(String[] args)  
{  
    new test13();  
}  
}
```

上面程序代码的运行结果如图 14.21 所示。



姓名	语文	数学	总分
王鹏	91	100	191
朱雪莲	82		182
梅庭	47	100	147
赵龙	61	100	161
李兵	90	100	190

图 14.21 自定义渲染器

这样就可以在一个大的表格中添加小的表格了，当然还有很多实际开发中用到的自定义渲染器，这里不再一一举例了。

14.9 如何为单元格指定文字说明

如何为单元格指定说明性文字？也就是说当鼠标指到单元格时，会出现一些说明性的文字。下面将通过一个实例讲解如何为单元格添加文字说明，其具体程序代码如下所示：

```
// 这段代码主要为读者展示如何为单元格指定说明性文字  
import java.awt.BorderLayout;  
import javax.swing.DefaultCellEditor;  
import javax.swing.JFrame;  
import javax.swing.JScrollPane;  
import javax.swing.JTable;  
import javax.swing.JTextField;  
import javax.swing.table.DefaultTableCellRenderer;  
import javax.swing.table.DefaultTableModel;  
import javax.swing.table.TableColumn;  
import javax.swing.table.TableColumnModel;  
public class test14 extends JFrame  
{  
    DefaultCellEditor cellEditor;  
    public test14()  
    {  
        super("输入文本测试");  
        String value[][]=  
        {  
            {"1","2","3","4","5"}  
        };  
        String columnName[]={"测试一","测试二","测试三","测试四","测试五"};  
        DefaultTableModel model=new DefaultTableModel(value,columnName);  
        JTable table=new JTable(model);  
        DefaultTableCellRenderer renderer=new DefaultTableCellRenderer();  
        renderer.setTooltipText("这是重要的数据信息");  
        TableColumnModel colmodel=table.getColumnModel();  
        // 为单元格指定说明性文字
```



```
for(int index=0;index<columnName.length;index++)
{
    TableColumn tc = colmodel.getColumn(index);
    tc.setCellRenderer(renderer);
}
this.add(new JScrollPane(table),BorderLayout.CENTER);
this.setSize(300, 200);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setVisible(true);
}
public static void main(String args[])
{
    new test14();
}
```

上面程序代码的运行结果如图 14.22 所示。

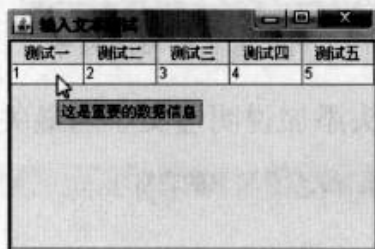


图 14.22 为单元格添加说明性文字

14.10 如何为表头指定文字说明

如何为表头指定说明性文字呢？也就是说当鼠标指到表头时，会出现说明性的文字。其实方法很简单，下面将通过实例来讲解如何为表头指定说明性文字。

```
// 这段代码主要是为读者展示如何为表头指定说明性文字
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
public class test15
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("JTable 的排序测试");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 表格中显示的数据
        Object rows[][]=
        {
            {"王鹏","江西","43"},
            {"周丹","四川","25"},
            {"钱丽","贵州","31"},
            {"孙军","江苏","24"},
            {"李环","新疆","45"},
            {"苏菲","广东","33"}
        };
        String columns[]={"姓名","籍贯","年龄"};
        TableModel model = new DefaultTableModel(rows, columns);
        Table table = new JTable(model);
```



```

        RowSorter sorter = new TableRowSorter(model);
        table.setRowSorter(sorter);
        JScrollPane pane = new JScrollPane(table);
        frame.add(pane, BorderLayout.CENTER);
        table.getTableHeader().setToolTipText("这个是有关学生信息的表格");
        // 为表头指定说明性文字
        frame.setSize(300, 150);
        frame.setVisible(true);
    }
}
    
```

上面程序段代码的运行结果如图 14.23 所示。

姓名	性别	年龄
王鹏	江西	43
周丹	四川	25
钱丽	浙江	31
孙军	江苏	24
李环	新疆	45
苏菲	广东	33

图 14.23 表头的说明性文字

通过以上实例可以看出，为表头添加说明性文字的最关键代码如下：

```
table.getTableHeader().setToolTipText("这个是有关学生信息的表格");
```

14.11 如何使用排序和过滤

排序和过滤基于一个新概念——行排序器对象，它能够对行数据进行排序（和过滤）。把一个行排序器加入到一个表格组件中的最简单的方法是调用 javax.swing.JTable 中新引入的 public void setAutoCreateRowSorter(boolean autoCreateRowSorter)，下面的代码片段演示了它的用法：

```

TableModel model = new DefaultTableModel(rows, columns);
JTable table = new JTable(model);
RowSorter sorter = new TableRowSorter(model);
table.setRowSorter(sorter);           // 排序和过滤的方法
    
```

希望读者能够记住：排序和过滤仅对视图有影响，而对模型无影响。下面将通过一个实例展示如何对表格数据进行排序和过滤。其实例代码如下所示：

```

// 这段代码主要是为读者展示如何将表格中的数据按大小进行排列
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
public class test16
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("JTable 的排序测试");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Object rows[][]=
        {
            {"王鹏","江西","43"},
            {"周丹","四川","25"},
            {"钱丽","浙江","31"},
            {"孙军","江苏","24"},
            {"李环","新疆","45"},
            {"苏菲","广东","33"}
        };
        // 表格中显示的数据
    }
}
    
```



```

        {"钱丽","贵州","31"},
        {"孙军","江苏","24"},
        {"李环","新疆","45"},
        {"苏菲","广东","33"}
    };
    String columns[]={"姓名","籍贯","年龄"};
    TableModel model = new DefaultTableModel(rows, columns);
    JTable table = new JTable(model);
    RowSorter sorter = new TableRowSorter(model);
    table.setRowSorter(sorter);// 表格排序过滤
    JScrollPane pane = new JScrollPane(table);
    frame.add(pane, BorderLayout.CENTER);
    frame.setSize(300, 150);
    frame.setVisible(true);
}
}

```

上面程序段代码的运行结果如图 14.24 所示。



姓名	籍贯	年龄
王鹏	江西	43
周丹	四川	25
钱丽	贵州	31
孙军	江苏	24
李环	新疆	45
苏菲	广东	33

图 14.24 过滤与排序

当双击“姓名”列时，会自动按姓氏首字母的升序或者降序来排列，效果如图 14.25 所示。



姓名 ▲	籍贯	年龄
李环	新疆	45
钱丽	贵州	31
苏菲	广东	33
孙军	江苏	24
王鹏	江西	43
周丹	四川	25

图 14.25 按姓名排序

当双击“籍贯”列时，会自动按籍贯首字母的升序或者降序来排列，效果如图 14.26 所示。



姓名	籍贯 ▲	年龄
苏菲	广东	33
钱丽	贵州	31
孙军	江苏	24
王鹏	江西	43
周丹	四川	25
李环	新疆	45

图 14.26 按籍贯排序

当双击“年龄”列时，会自动按年龄数字的升序或者降序来排列，效果如图 14.27 所示。



姓名	籍贯	年龄 ▲
孙军	江苏	24
周丹	四川	25
钱丽	贵州	31
苏菲	广东	33
王鹏	江西	43
李环	新疆	45

图 14.27 按年龄排序

以上的实例很简单，但是它展示了排序的最基本的使用方法。可以仿照上例，自行列举一些比较复杂的实例来练习。

14.12 如何使用组合框作为编辑器

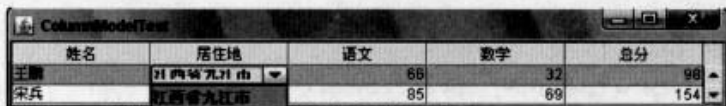
在实际开发中，很多表格中的单元格都是以组合框作为编辑器的，下面将通过实例进行说明，其程序代码如下所示：

```
// 这段代码主要是为读者展示如何使用组合列表框作为表格中的元素
// 使得可以在列表框中选择所需要的数据，相当于输入不同数据
import javax.swing.table.AbstractTableModel;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test17
{
    public test17()
    {
        JFrame f = new JFrame();
        /* 由于 MyTable 类继承了 AbstractTableModel
        * 并且实现了 getColumnCount()、getRowCount()、getValueAt()方法。因此可以通
        * 过 MyTable 来产生 TableModel 的实体
        */
        MyTable mt = new MyTable();
        JTable t = new JTable(mt);           // 利用 MyTable 来建立 JTable
        JComboBox c = new JComboBox();      // 建立一个 JComboBox 的对象
        c.addItem("江西省九江市");         // 在新建的 JComboBox 对象里新增三个项目
        c.addItem("浙江省杭州市");
        c.addItem("安徽省合肥市");
        /*利用 JTable 所提供的 getTableColumnModel()方法取得 TableColumnModel 对象
        * 再由 TableColumnModel 类所提供的 getColumn()方法取得 TableColumn 对象
        * TableColumn 类可针对表格中的每一行做具体的设置
        * 例如设置字段的宽度、某行的标头、设置输入较复杂的数据类型等
        * 在这里利用 TableColumn 类所提供的 setCellEditor()方法，将 JComboBox 作为第 2 行的默认编辑组件
        */
        t.getColumnModel().getColumn(1).setCellEditor(new DefaultCellEditor(c));
        t.setPreferredSize(new Dimension(550, 30));
        JScrollPane s = new JScrollPane(t);
        f.getContentPane().add(s, BorderLayout.CENTER);
        f.setTitle("ColumnModelDd");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String args[])
    {
        new test17();
    }
}
// 继承表格抽象类的表格模型
class MyTable extends AbstractTableModel
{
```



```
Object[][] p =
{
    {"王鹏", "江西省九江市", new Integer(66), new Integer(32), new Integer(98)},
    {"宋兵", "浙江省杭州市", new Integer(85), new Integer(69), new Integer(154)},
};
String[] n =
{ "姓名", "居住地", "语文", "数学", "总分", };
public int getColumnCount()
{
    return n.length;
}
public int getRowCount()
{
    return p.length;
}
public String getColumnName(int col)
{
    return n[col];
}
public Object getValueAt(int row, int col)
{
    return p[row][col];
}
public Class getColumnClass(int c)
{
    return getValueAt(0, c).getClass();
}
public boolean isCellEditable(int rowIndex, int columnIndex)
{
    return true;
}
public void setValueAt(Object value, int row, int col)
{
    p[row][col] = value;
    fireTableCellUpdated(row, col);
}
}
```

上面程序段代码的运行结果如图 14.28 所示。



姓名	居住地	语文	数学	总分
王鹏	江西省九江市	66	32	98
宋兵	浙江省杭州市	85	69	154

图 14.28 添加组合框作为编辑器

上面将组合列表框作为表格单元中的一个元素来编辑，这样增加了程序中表格的灵活性。在实际开发中，还有很多组件可以用作此途，这里不再详细介绍。

14.13 如何使用其他编辑器

本节将继续上节的内容，使用其他的组件作为编辑器编辑单元格。下面仍通过实例讲解其他的组件如何用作编辑器，其程序代码如下所示：


```
// 这段代码主要是为读者展示如何使用复选框作为编辑器来编辑表格中的单元格
import javax.swing.table.AbstractTableModel;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class test18
{
    public test18()
    {
        JFrame f = new JFrame();
        MyTable mt=new MyTable();
        /* 由于 MyTable 类继承了 AbstractTableModel, 并且实现了 getColumnCount()、getRowCount()、getValueAt()方法
        * 因此可以通过 MyTable 来产生 TableModel 的实体
        */
        JTable t=new JTable(mt);// 利用 MyTable 来建立 JTable
        JCheckBox jc1=new JCheckBox();
        t.getColumnModel().getColumn(4).setCellEditor(new DefaultCellEditor(jc1));
        t.setPreferredSize(new Dimension(550, 30));
        JScrollPane s = new JScrollPane(t);
        f.getContentPane().add(s, BorderLayout.CENTER);
        f.setTitle("ColumnModelDd");
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String args[])
    {
        new test18();
    }
}

class MyTable extends AbstractTableModel
{
    Object[][] p =
    {
        {"王鹏", new Integer(66),new Integer(32), new Integer(98),""},
        {"宋兵", new Integer(85),new Integer(69), new Integer(154),""},
    };
    String[] n = {"姓名","语文","数学","总分","及格与否"};
    public int getColumnCount()
    {
        return n.length;
    }
    public int getRowCount()
    {
        return p.length;
    }
    public String getColumnName(int col)
    {
        return n[col];
    }
}
```



```

public Object getValueAt(int row, int col)
{
    return p[row][col];
}
public Class getColumnClass(int c)
{
    return getValueAt(0, c).getClass();
}
public boolean isCellEditable(int rowIndex, int columnIndex)
{
    return true;
}
public void setValueAt(Object value, int row, int col)
{
    p[row][col] = value;
    fireTableCellUpdated(row, col);
}
}

```

上面程序代码的运行结果如图 14.29 所示。

姓名	语文	数学	总分	及格与否
王刚	66	32	98	<input type="checkbox"/>
宋兵	85	69	154	<input checked="" type="checkbox"/>

图 14.29 使用 JCheckBox 组件作为单元格编辑器

其实，除了可以使用 JCheckBox、组合列表框组件作为单元格的编辑器外，还可以使用其他的很多组件，这里不再一一举例了，读者可以自行查阅相关资料，多多练习。

14.14 如何使用编辑器验证文本

在实际开发中，开发人员会考虑一点，即当用户输入的信息不属于该范围，例如需要输入年龄，用户错误地输入了家庭地址，此时编辑器会采取一定的措施。那么如何来实现这种功能呢？下面将列举一个简单的实例来介绍如何使用编辑器来响应用户输入的文本内容。其程序代码如下所示：

```

// 这段代码主要是展示如何使用编辑器来验证用户输入的数据的正确与否
import java.awt.BorderLayout;
import java.awt.Component;
import javax.swing.DefaultCellEditor;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
import javax.swing.table.TableColumnModel;
public class test19 extends JFrame
{
    public test19()
    {
        super("输入文本测试");
        String value[][] = {"1", "2", "3", "4", "5"};
    }
}

```



```
String columnName[]={"测试一","测试二","测试三","测试四","测试五"};
DefaultTableModel model=new DefaultTableModel(value,columnName);
JTable table=new JTable(model);
TableColumnModel colmodel=table.getColumnModel();
// 这个 for 循环语句设置一个文本框组件，将它添加到表格的单元格
// 并且设置单元格单击可编辑
for(int index=0;index<columnName.length;index++)
{
    TableColumn tc = colmodel.getColumn(index);
    JTextField editor=new JTextField();
    MyCellEditor cellEditor=new MyCellEditor(editor);
    cellEditor.setClickCountToStart(1);
    tc.setCellEditor(cellEditor);
}
this.add(new JScrollPane(table),BorderLayout.CENTER);
this.setSize(300, 200);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setVisible(true);
}
public static void main(String args[])
{
    new test19();
}
}
class MyCellEditor extends DefaultCellEditor
{
    public MyCellEditor(JTextField textField)
    {
        super(textField);
    }
    // 可以通过重载这个方法设置单元格的属性
    public boolean stopCellEditing()
    {
        // 当在单元格中输入“1234”时，单元格清空
        String value=(String)this.getCellEditorValue();
        if(value.equals("1234"))
        {
            ((JTextField)this.getComponent()).setText("");
            return false;
        }
        else
            return super.stopCellEditing();
    }
}
}
```

上面程序代码的运行结果如图 14.30 所示。

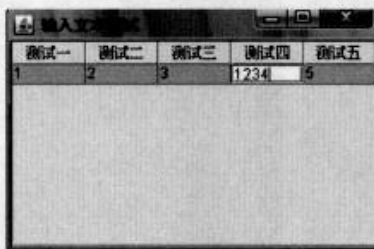


图 14.30 利用编辑器响应用户文本输入 (a)

当在编辑器中输入“1234”非法数据时，编辑器会自动清空数据，如图 14.31 所示。

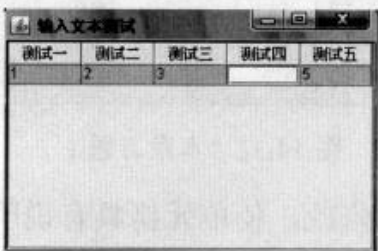


图 14.31 利用编辑器响应用户文本输入 (b)

14.15 如何打印表格

在 Swing 中打印表格很简单,共有 4 种方法，如表 14.8 所示。

表 14.8 打印表格的方法

方法	说明
<code>print()</code>	它显示一个打印对话框,然后以 <code>PrintMode.FIT_WIDTH</code> 模式打印此 <code>JTable</code> , 不打印标题或脚注文本
<code>print(JTable.PrintMode printMode)</code>	它显示一个打印对话框,然后以给定的打印模式打印此 <code>JTable</code> , 不打印标题或脚注文本
<code>print(JTable.PrintMode printMode,MessageFormat headerFormat, MessageFormat footerFormat)</code>	它显示一个打印对话框,然后以给定的打印模式打印此 <code>JTable</code> , 打印指定的标题和脚注文本
<code>print(JTable.PrintMode printMode,MessageFormat headerFormat,MessageFormat footerFormat, boolean showPrintDialog,PrintRequestAttributeSet attr, boolean interactive)</code>	打印此 <code>JTable</code>

14.16 本章小结

本章主要介绍了有关表格的建立、使用以及在实际开发中常用的技巧。在讲述表格的建立时，使用了三种方法，分别是 `JTable()`、`AbstractTableModel()`、`DefaultTableModel()`。另外，还讲述了有关单元格编辑器和渲染器的知识以及其应用，它们是在实际开发中应用非常广泛的知识点。当然，在实际开发中，还有很多相关的知识点没有在本书中提到，希望读者能够以本章的内容为基础，自行查阅资料。

14.17 本章习题

1. 设计一个简单表格，表格中的列有姓名、籍贯、年龄、出生年月、家庭地址、所在公司。
- 要求：其最终效果如图 14.32 所示。



姓名	籍贯	年龄	出生年月	家庭地址	所在公司
王鹏	江西九江	32	1975	上海浦东	上海贝尔
朱雪莲	上海浦东	27	1981	上海浦东	上海大众
梅庭	江苏启东	28	1980	上海徐汇	上海通用
赵龙	黑龙江哈...	33	1974	上海长宁	上海Intel
李兵	浙江杭州	34	1974	上海松江	上海中信

图 14.32 本章习题 1

2. 将本章习题 1 中的表格进行修改, 使单元格具有说明性文字。

要求:

(1) 程序运行后, 当将鼠标放在“姓名”列后, 将会出现说明性文字, 其效果如图 14.33 所示。



姓名	籍贯	年龄	出生年月	家庭地址	所在公司
王鹏	江西九江	32	1975	上海浦东	上海贝尔
朱雪莲	上海浦东	27	1981	上海浦东	上海大众
梅庭	江苏启东	28	1980	上海徐汇	上海通用
赵龙	黑龙江哈...	33	1974	上海长宁	上海Intel
李兵	浙江杭州	34	1974	上海松江	上海中信

图 14.33 本章习题 2 (a)

(2) 当将鼠标放在“年龄”列后, 将会出现说明性文字, 其效果如图 14.34 所示。



姓名	籍贯	年龄	出生年月	家庭地址	所在公司
王鹏	江西九江	32	1975	上海浦东	上海贝尔
朱雪莲	上海浦东	27	1981	上海浦东	上海大众
梅庭	江苏启东	28	1980	上海徐汇	上海通用
赵龙	黑龙江哈...	33	1974	上海长宁	上海Intel
李兵	浙江杭州	34	1974	上海松江	上海中信

图 14.34 本章习题 2 (b)

3. 将本章习题 1 中的表格进行修改, 使此表格可以通过双击表头而进行自动排序。

要求: 双击任何一个表头列, 都会自动对其列中的数据进行排列。

4. 将本章习题 1 中的表格进行修改, 将表格打印出来。

要求: 程序运行结果如图 14.35 所示。



姓名	籍贯	年龄	出生年月	家庭地址	所在公司
王鹏	江西九江	32	1975	上海浦东	上海贝尔
朱雪莲	上海浦东	27	1981	上海浦东	上海大众
梅庭	江苏启东	28	1980	上海徐汇	上海通用
赵龙	黑龙江哈...	33	1974	上海长宁	上海Intel
李兵	浙江杭州	34	1974	上海松江	上海中信

打印表格

图 14.35 本章习题 4

第15章 如何使用树组件

什么是树呢？提起树，读者一定会联想到生活中的参天大树，它有树根、树枝、树叶等。在程序开发中的树是一种组件，它也有根、枝、叶。不知道读者是否还记得目录结构，目录结构就是一个树组件，例如：“c:/tree/tee.txt”，其中“c:/”就是这个目录的树根，tree 就是这个目录的树枝，而 tree.txt 就是这个目录的树叶。这样理解是否觉得容易多了呢？在下面的章节中将会为读者详细介绍有关树的一些基本概念、创建方法以及在实际开发中如何更好地应用它等。

15.1 如何创建树

为了能够更加形象的使读者清楚什么是树，以及目录为什么是树结构，下面将给出目录结构图，如图 15.1 所示。

树组件在 Swing 库中就是 JTree，在学习如何创建树之前，先为读者列出有关树组件的继承结构图，如图 15.2 所示。

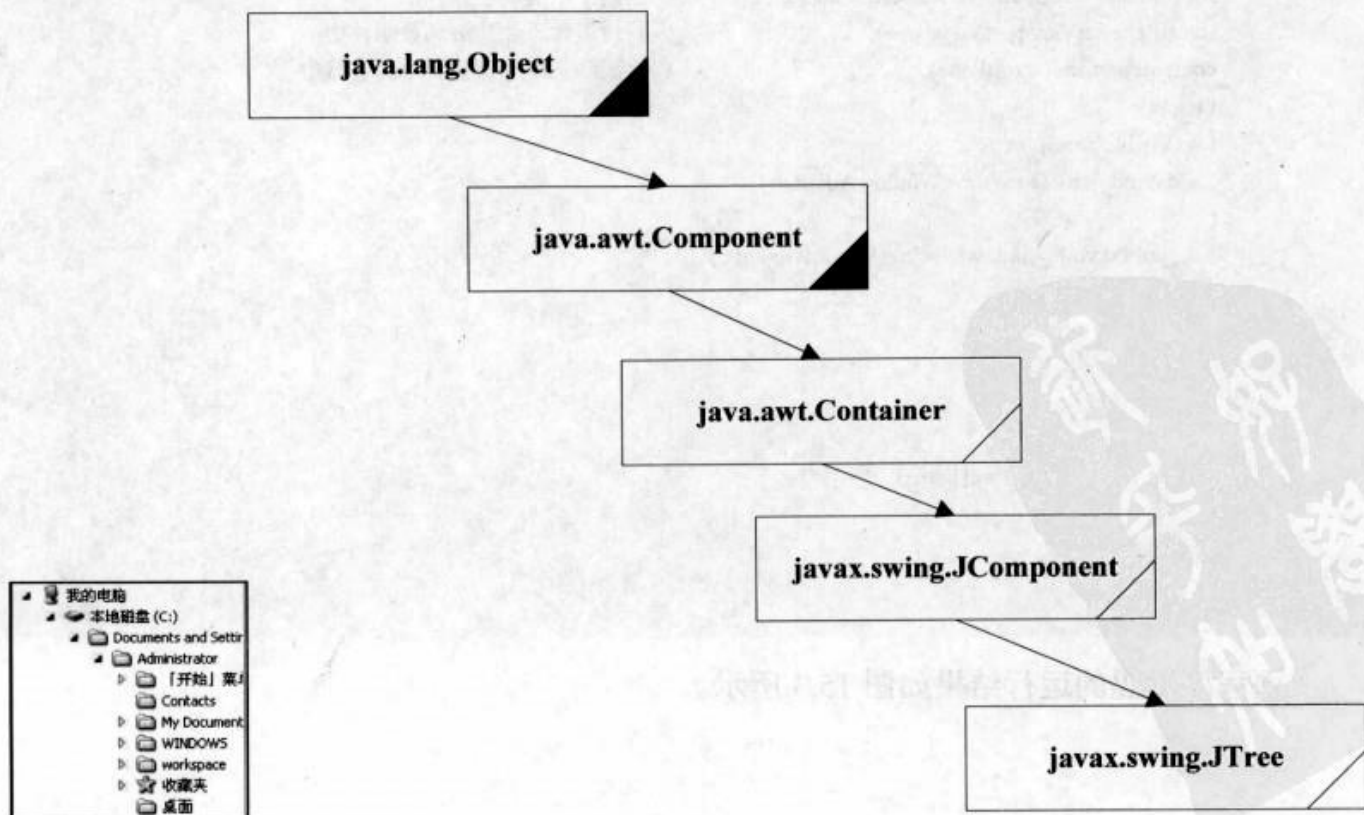


图 15.1 目录结构图

图 15.2 树组件继承结构图

从上面的继承结构图可以知道,树组件继承自 JComponent 类,所以它不能独立显示,必须依赖于顶层容器才能显示。下面将为读者介绍 JTree 的构造器,如表 15.1 所示。

表 15.1 JTree 构造器

JTree 构造器	说明
JTree()	建立一棵系统默认的树
JTree(Hashtable value)	利用 Hashtable 建立树,不显示 root node 根节点
JTree(Object[] value)	利用 Object Array 建立树,不显示 root node 根节点
JTree(TreeModel newModel)	利用 TreeModel 建立树
JTree(TreeNode root)	利用 TreeNode 建立树
JTree(TreeNode root,boolean asksAllowsChildren)	利用 TreeNode 建立树,并决定是否允许子节点的存在
JTree(Vector value)	利用 Vector 建立树,不显示 root node

上面的表格介绍了树组件的构造器,接下来将通过一个简单程序实例讲述如何创建 JTree 组件。这个实例主要是通过构造器 JTree()创建一个树组件,其具体的实例程序代码如下所示:

```
// 这段程序代码主要是为读者展示如何通过 JTree()来创建树组件
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test1
{
    public test1()
    {
        JFrame f=new JFrame("树组件测试");
        Container contentPane=f.getContentPane();
        JTree tree=new JTree();           // 创建一个系统默认的树组件
        JScrollPane scrollPane=new JScrollPane(); // 创建一个滚动条组件
        scrollPane.setViewportView(tree);       // 将树组件添加到滚动条组件中
        contentPane.add(scrollPane);       // 将滚动条组件添加到中间容器中
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        new test1();
    }
}
```

上面程序代码的运行结果如图 15.3 所示。

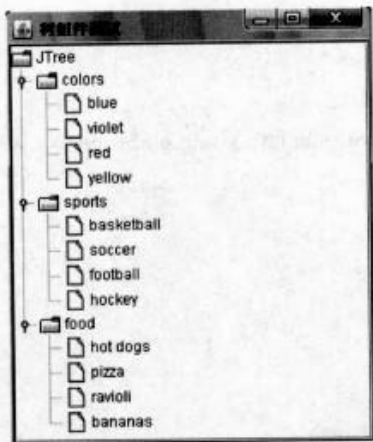


图 15.3 创建一个默认的树组件

上面的程序创建了一个系统默认的树组件，即当使用不带参数的构造器创建树时，会出现如上图所示的结构。

下面再给出一个实例程序代码，这个实例将会通过构造器 `JTree(Hashtable value)` 来创建一个树组件，其中 `Hashtable` 表是一种作为数据存储的数据结构，希望读者能够记住，使用这种方法创建树组件是不会显示其根节点的。其具体的程序代码如下所示：

// 这段程序代码主要是为读者展示如何使用哈希表创建树组件

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class test2
{
    public test2()
    {
        JFrame f=new JFrame("树组件测试");
        Container contentPane=f.getContentPane();
        String[] s1={"青菜","大蒜","大葱"}; // 创建三个字符串数组
        String[] s2={"苹果","梨子","香蕉"};
        String[] s3={"馒头","包子","饺子","馄饨","面条"};
        Hashtable hashtable1=new Hashtable(); // 创建两个哈希表对象
        Hashtable hashtable2=new Hashtable();
        hashtable1.put("蔬菜",s1);
        /* 将 s1 添加到哈希表对象 1 中，并且将字符串数组命名为蔬菜，将 s2 添加到哈希表对象 1 中
        * 并且将字符串数组命名为水果，将哈希表对象 1 添加到哈希表对象 2 中，并且给字符串数组命名为点心
        * 将 s3 添加到哈希表对象 2 中，并且将字符串数组命名为中点
        */
        hashtable1.put("水果",s2);
        hashtable1.put("点心",hashtable2);
        hashtable2.put("中点",s3);
        Font font = new Font("Dialog", Font.PLAIN, 12);
        Enumeration keys = UIManager.getLookAndFeelDefaults().keys();
        /**定义 Windows 界面**/
        while (keys.hasMoreElements())
        {
            Object key = keys.nextElement();
            if (UIManager.get(key) instanceof Font)
            {
                UIManager.put(key, font);
            }
        }
    }
}
```



```
}
Try
{
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");// 设置界面的观感器
}
catch(Exception e1)
{
    System.exit(0);
}
/**定义 Windows 界面**/
JTree tree=new JTree(hashtable1);
JScrollPane scrollPane=new JScrollPane();
scrollPane.setViewportView(tree);
contentPane.add(scrollPane);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    new test2();
}
}
```

上面实例程序代码的运行结果如图 15.4 所示。

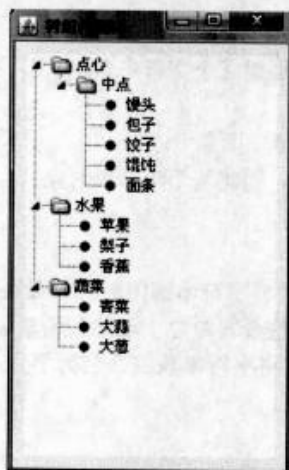


图 15.4 使用哈希表创建树组件

上面的实例比较简单，惟一需要说明的就是 `UIManager.getLookAndFeelDefaults()` 的含义，它的意义就是让树结构呈现出默认的观感器，至于观感器的知识将会在后面的章节中单独进行讲述。其实，使用哈希表构建树组件还不是最好的方式，在后面的章节中会再为读者介绍另一种构建树组件的方法——以 `TreeNode` 方式构造 `JTree`。

15.2 如何创建数据模型

JTree 上的每一个节点就代表一个 `TreeNode` 对象，`TreeNode` 本身是一个接口，里面定义了 7 个有关节点的方法，其中包括判断是否为叶节点、有几个子节点 (`getChildCount()`)、父节点是什么 (`getParent()`) 等。在实际开发中，一般不会直接实现此接口，而是采用 Java 所提供的 `DefaultMutableTreeNode` 类，此类用于实现 `MutableTreeNode` 接口，并提供其他许多实用的方法。`MutableTreeNode` 本身也是一个接口，并且继承了 `TreeNode` 接口，此类主要是定义一些节点的处理方式，如添加节点 (`insert()`)、删除节点 (`remove()`)、设置节点 (`setUserObject()`) 等。

下面将通过 15.2 列举出 `DefaultMutableTreeNode` 的构造器。

表 15.2 `DefaultMutableTreeNode` 的构造器

DefaultMutableTreeNode 的构造器	说明
<code>DefaultMutableTreeNode()</code>	建立空的 <code>DefaultMutableTreeNode</code> 对象
<code>DefaultMutableTreeNode(Object userObject)</code>	建立 <code>DefaultMutableTreeNode</code> 对象，节点为 <code>userObject</code> 对象
<code>DefaultMutableTreeNode(Object userObject, Boolean allowsChildren)</code>	建立 <code>DefaultMutableTreeNode</code> 对象，节点为 <code>userObject</code> 对象，并决定此节点是否允许具有子节点

下面将针对上面表格所列举出来的构造器列举一个实例，这个实例主要是使用 `DefaultMutableTreeNode` 的构造器来创建树组件中的每一个节点，最后形成一棵树，其具体的程序代码如下所示：

```
/*这段程序代码主要是展示如何使用 DefaultMutableTreeNode 的构造器来创建树组件中的每一个节点，最后形成一棵树*/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
public class test3
{
    public test3()
    {
        JFrame f=new JFrame("今天要购买的清单");
        Container contentPane=f.getContentPane();
        DefaultMutableTreeNode root=new DefaultMutableTreeNode("今天要购买的东西");
        // 使用 DefaultMutableTreeNode 的构造器创建根节点
        DefaultMutableTreeNode node1=new DefaultMutableTreeNode("蔬菜");
        // 使用 DefaultMutableTreeNode 的构造器创建 4 个枝节点
        DefaultMutableTreeNode node2=new DefaultMutableTreeNode("水果");
        DefaultMutableTreeNode node3=new DefaultMutableTreeNode("礼品");
        DefaultMutableTreeNode node4=new DefaultMutableTreeNode("家用小物件");
        root.add(node1); // 将 4 个枝节点添加到根节点中
        root.add(node2);
        root.add(node3);
        root.add(node4);
        DefaultMutableTreeNode leafnode=new DefaultMutableTreeNode("白菜");
        // 利用 DefaultMutableTreeNode 的构造器创建叶节点，再将叶节点分别添加到不同的枝节点上
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("大蒜");
        node1.add(leafnode);
    }
}
```



```
leafnode=new DefaultMutableTreeNode("土豆");
node1.add(leafnode);
leafnode=new DefaultMutableTreeNode("苹果");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("香蕉");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("西瓜");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("礼品");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("茅台酒");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("营养麦片");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("保健食品");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("味精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("酱油");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("洗洁精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("保险袋");
node4.add(leafnode);
JTree tree=new JTree(root);
JScrollPane scrollPane=new JScrollPane();
scrollPane.setViewportView(tree);
contentPane.add(scrollPane);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    new test3();
}
```

上面程序代码的运行结果如图 15.5 所示。

使用 `TreeNode` 创建树组件的原理就是使用 `DefaultMutableTreeNode` 的构造器创建树中的每一个节点，然后将叶节点添加到枝节点中，再将枝节点添加到根节点中。

在实际的软件开发中，不可能只开发一个树组件，而且要让每个节点的细节都要呈现在用户面前，也就是说当单击任何一个节点时，就会出现相应的细节。这就涉及到了响应节点选择事件。在下一节中将为读者讲述如何响应节点的事件。

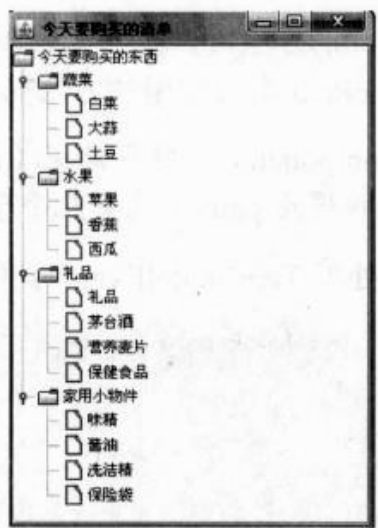


图 15.5 使用 TreeNode 创建树组件

15.3 如何处理节点事件

本节将详细介绍 JTree 中两个常用的事件与处理方法，分别是 TreeModelEvent 与 TreeSelectionEvent。当树的结构有任何改变时，例如节点值改变了、新增节点、删除节点等，都会触发 TreeModelEvent 事件。当选择任何一个节点时，都会触发 TreeSelectionEvent 事件。下面将通过两个小节的内容向读者介绍如何处理这些事件。

15.3.1 如何处理 TreeModelEvent 事件

要处理 TreeModelEvent 事件，就必须实现 TreeModelListener 接口，此接口定义了 4 个方法，如表 15.3 所示。

表 15.3 TreeModelListener 接口方法

方法	说明
Void treeNodesChanged(TreeModelEvent e)	当节点改变时系统就会调用这个方法
Void treeNodesInserted(TreeModelEvent e)	当新增节点时系统就会调用这个方法
Void treeNodesRemoved(TreeModelEvent e)	当删除节点时系统就会调用这个方法
Void treeStructureChanged(TreeModelEvent e)	当树结构改变时系统就会调用这个方法

由于要实现 TreeModelListener 接口，所以就必须要实现上面 4 个接口方法。其实 TreeModelEvent 类本身提供了 5 个方法，用来获取树组件信息，如表 15.4 所示。

表 15.4 TreeModelEvent 的方法

方法	说明
int[] getChildIndices()	返回子节点群的索引值
Object[] getChildren()	返回子节点群
Object[] getPath()	返回 Tree 中一条 path 上（从 root node 到 leaf node）的节点
TreePath getTreePath()	取得目前位置的 Tree Path
String toString()	取得字符串表示法

由 `TreeModelEvent` 的 `getTreePath()` 方法可以得到 `TreePath` 对象, 此对象能够让系统知道用户目前正在选择哪一个节点, `TreePath` 类最常用的方法如下:

- `public Object getLastPathComponent()`: 取得最深(内)层的节点。
- `public int getPathCount()`: 取得此 path 上共有几个节点。

下面将通过一个实例讲解如何处理 `TreeModelEvent` 事件。其程序代码如下所示:

```
// 这段程序代码主要是为读者展示如何处理 TreeModelEvent 事件
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.*;

public class test4 implements TreeModelListener
{
    JLabel label;
    String nodeName = null; // 原有节点名称

    public test4()
    {
        JFrame f=new JFrame("今天要购买的清单");
        Container contentPane=f.getContentPane();
        DefaultMutableTreeNode root=new DefaultMutableTreeNode("今天要购买的东西");
        DefaultMutableTreeNode node1=new DefaultMutableTreeNode("蔬菜");
        DefaultMutableTreeNode node2=new DefaultMutableTreeNode("水果");
        DefaultMutableTreeNode node3=new DefaultMutableTreeNode("礼品");
        DefaultMutableTreeNode node4=new DefaultMutableTreeNode("家用小物件");
        root.add(node1);
        root.add(node2);
        root.add(node3);
        root.add(node4);
        DefaultMutableTreeNode leafnode=new DefaultMutableTreeNode("白菜");
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("大蒜");
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("土豆");
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("苹果");
        node2.add(leafnode);
        leafnode=new DefaultMutableTreeNode("香蕉");
        node2.add(leafnode);
        leafnode=new DefaultMutableTreeNode("西瓜");
        node2.add(leafnode);
        leafnode=new DefaultMutableTreeNode("礼品");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("茅台酒");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("营养麦片");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("保健食品");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("味精");
```



```

node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("酱油");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("洗洁精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("保险袋");
node4.add(leafnode);
JTree tree=new JTree(root);
tree.setEditable(true);// 设置 JTree 为可编辑的
tree.addMouseListener(new MouseHandle());
// 在 Tree 中加入检测 Mouse 事件, 以便取得节点名称
DefaultTreeModel treeModel = (DefaultTreeModel)tree.getModel();
// 下面两行取得 DefaultTreeModel, 并检测是否有 TreeModelEvent 事件
treeModel.addTreeModelListener(this);
JScrollPane scrollPane = new JScrollPane();
scrollPane.setViewPortView(tree);
label = new JLabel("更改数据为: ");
contentPane.add(scrollPane,BorderLayout.CENTER);
contentPane.add(label,BorderLayout.SOUTH);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public void treeNodesChanged(TreeModelEvent e)
{
    /* 本方法实现了 TreeModelListener 接口, 本接口共定义 4 个方法, 分别是 TreeNodesChanged()
    * treeNodesInserted()、treeNodesRemoved()、treeNodesRemoved()、treeStructureChanged()
    * 在此范例中只针对更改节点值的部分, 因此只须实现 treeNodesChanged()方法
    */
    TreePath treePath = e.getTreePath();// 获取目前所选取节点的树路径
    DefaultMutableTreeNode node = (DefaultMutableTreeNode)treePath.getLastPathComponent();
    // 获得所选取节点
    try
    {
        // 获取其子节点的节点序号, 针对此序号获得其子节点
        int[] index = e.getChildIndices();
        node = (DefaultMutableTreeNode)node.getChildAt(index[0]);
    }
    catch (NullPointerException exc) {}
    label.setText(nodeName+"更改数据为: "+(String)node.getUserObject());
    // 将节点内容显示在标签中
}
public void treeNodesInserted(TreeModelEvent e) {}
public void treeNodesRemoved(TreeModelEvent e) {}
public void treeStructureChanged(TreeModelEvent e) {}
public static void main(String args[])
{
    new test4();
}
class MouseHandle extends MouseAdapter
{
    // 处理 Mouse 点选事件
    public void mousePressed(MouseEvent e)

```



```

{
    Try
    {
        JTree tree = (JTree)e.getSource();// 当单击树中的节点时，会获取节点的内容
        int rowLocation = tree.getRowForLocation(e.getX(), e.getY());
        TreePath treepath = tree.getPathForRow(rowLocation);
        TreeNode treenode = (TreeNode) treepath.getLastPathComponent();
        nodeName = treenode.toString();
    }
    catch(NullPointerException ne){}
}
}
}

```

上面程序代码的运行结果如图 15.6 所示。



图 15.6 TreeModelEvent 事件处理 (a)

当修改一个节点后，其结果如图 15.7 所示。



图 15.7 TreeModelEvent 事件处理 (b)

上面的程序中当修改“白菜”为“洋葱”时，就会触发 TreeModelEvent 事件，于是会将修改后的内容以标签的形式给出。在实际开发中，以上实例的方法是经常使用的，它会提醒用户修改前后的内容变化。下面再给出一个稍微复杂一点的实例，其实例代码如下所示：


```
// 这段程序代码主要是展示如何处理 TreeModelEvent 事件
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.*;

public class test5 implements TreeModelListener
{
    JLabel label;
    String nodeName = null; // 原有节点名称
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode node1;
    DefaultMutableTreeNode node2;
    DefaultMutableTreeNode node3;
    DefaultMutableTreeNode node4;
    JTree tree;
    DefaultTreeModel treeModel;

    public test5()
    {
        JFrame f=new JFrame("今天要购买的清单");
        Container contentPane=f.getContentPane();
        Container con=new Container();
        root=new DefaultMutableTreeNode("今天要购买的东西");
        node1=new DefaultMutableTreeNode("蔬菜");
        node2=new DefaultMutableTreeNode("水果");
        node3=new DefaultMutableTreeNode("礼品");
        node4=new DefaultMutableTreeNode("家用小物件");
        root.add(node1);
        root.add(node2);
        root.add(node3);
        root.add(node4);
        DefaultMutableTreeNode leafnode=new DefaultMutableTreeNode("白菜");
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("大蒜");
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("土豆");
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("苹果");
        node2.add(leafnode);
        leafnode=new DefaultMutableTreeNode("香蕉");
        node2.add(leafnode);
        leafnode=new DefaultMutableTreeNode("西瓜");
        node2.add(leafnode);
        leafnode=new DefaultMutableTreeNode("礼品");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("茅台酒");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("营养麦片");
        node3.add(leafnode);
        leafnode=new DefaultMutableTreeNode("保健食品");
```



```

node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("味精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("酱油");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("洗洁精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("保险袋");
node4.add(leafnode);
tree=new JTree(root);
tree.setEditable(true);// 设置 JTree 为可编辑的
tree.addMouseListener(new MouseHandle());
// 使 Tree 加入检测 Mouse 事件, 以便取得节点名称
final DefaultTreeModel treeModel = (DefaultTreeModel)tree.getModel();
// 下面两行取得 DefaultTreeModel, 并检测是否有 TreeModelEvent 事件
treeModel.addTreeModelListener(this);
JScrollPane scrollPane = new JScrollPane();
scrollPane.setViewportView(tree);
label = new JLabel("更改数据为: ");
JButton b1=new JButton("增加节点");
JButton b2=new JButton("删除节点");
con.setLayout(new FlowLayout());
con.add(b1);
con.add(b2);
con.add(label);
contentPane.add(scrollPane,BorderLayout.CENTER);
contentPane.add(con,BorderLayout.SOUTH);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
...
}

```

以上代码创建了一个树组件的界面, 下面将针对按钮组件的动作事件进行处理, 其具体代码如下所示:

```

b1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        DefaultMutableTreeNode parentNode=null;
        DefaultMutableTreeNode newNode=new DefaultMutableTreeNode("新节点");
        newNode.setAllowsChildren(true);
        TreePath parentPath=tree.getSelectionPath();
        parentNode=(DefaultMutableTreeNode)(parentPath.getLastPathComponent());
        // 取得新节点的父节点
        treeModel.insertNodeInto(newNode,parentNode,parentNode.getChildCount());
        // 由 DefaultTreeModel 的 insertNodeInto()方法增加新节点
        tree.scrollPathToVisible(new TreePath(newNode.getPath()));
        // tree 的 scrollPathToVisible()方法使 Tree 会自动展开文件夹, 以便显示所加入的新节点
    }
}

```



```

// 若没加这行则加入的新节点会被包在文件夹中，必须自行展开文件夹才能看到
label.setText("新增节点成功");
}
});
b2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        TreePath treepath=tree.getSelectionPath();
        if (treepath!=null)
        {
            DefaultMutableTreeNode selectionNode=(DefaultMutableTreeNode)treepath.getLastPathComponent();
            // 下面两行用于选取节点的父节点
            TreeNode parent=(TreeNode)selectionNode.getParent();
            if (parent!=null)
            {
                treeModel.removeNodeFromParent(selectionNode);
                // 由 DefaultTreeModel 的 removeNodeFromParent()方法删除节点，包含它的子节点
                label.setText("删除节点成功");
            }
        }
    }
});

```

上面的代码针对按钮动作事件进行了处理，下面将实现有关树组件节点变化事件的处理，其具体代码如下所示：

```

/* 本方法实现了 TreeModelListener 接口，本接口共定义 4 个方法，分别是 TreeNodesChanged()
 * treeNodesInserted()、treeNodesRemoved()、treeNodesRemoved()、
 * treeStructureChanged()，在此范例中只针对更改节点值的部分，因此只实现了
 * treeNodesChanged()方法
 */
public void treeNodesChanged(TreeModelEvent e)
{
    TreePath treePath = e.getTreePath();
    DefaultMutableTreeNode node = (DefaultMutableTreeNode)treePath.getLastPathComponent();
    try
    {
        int[] index = e.getChildIndices();
        node = (DefaultMutableTreeNode)node.getChildAt(index[0]);
    }
    catch (NullPointerException exc) {}
    label.setText(nodeName+"更改数据为: "+(String)node.getUserObject());
}
public void treeNodesInserted(TreeModelEvent e)
{
}
public void treeNodesRemoved(TreeModelEvent e)
{
}
public void treeStructureChanged(TreeModelEvent e)
{
}
public static void main(String args[])
{
    new test5();
}

```

上面处理了节点变化事件，下面将给出有关鼠标双击后的事件处理代码：


```
class MouseHandle extends MouseAdapter
{
    // 处理 Mouse 点选事件
    public void mousePressed(MouseEvent e)
    {
        Try
        {
            JTree tree = (JTree)e.getSource();
            int rowLocation = tree.getRowForLocation(e.getX(), e.getY());
            TreePath treepath = tree.getPathForRow(rowLocation);
            TreeNode treenode = (TreeNode) treepath.getLastPathComponent();
            nodeName = treenode.toString();
        }
        catch (NullPointerException ne){}
    }
}
```

上面程序代码的运行结果如图 15.8 所示。



图 15.8 删除和添加树组件的节点

当单击“蔬菜”选项后单击“增加节点”按钮，将在“蔬菜”选项下多出一个新节点，并且标签会弹出“添加节点成功”的字样，如图 15.9 所示。

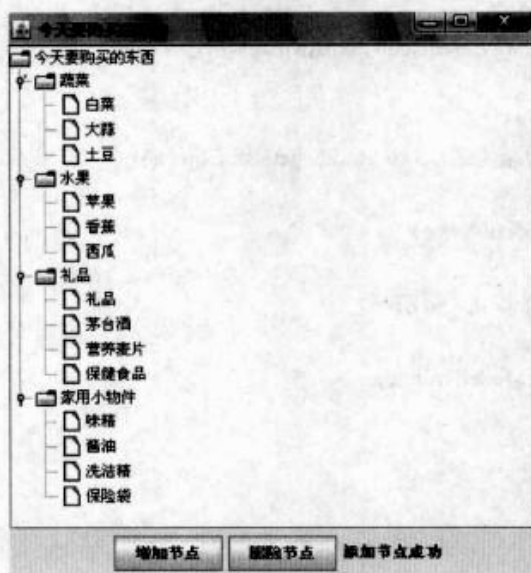


图 15.9 添加树组件的节点

当单击“水果”下的“苹果”选项后，单击“删除节点”按钮，“苹果”这个节点会被删除，并且标签会弹出“删除节点成功”的字样，如图 15.10 所示。

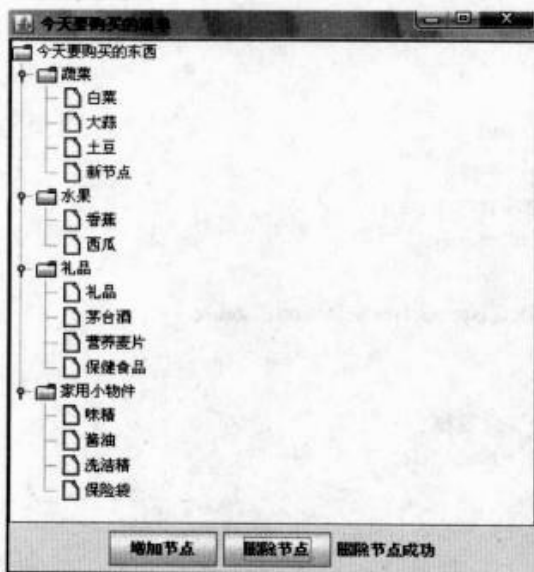


图 15.10 删除树组件的节点

以上实例演示了在树组件中如何添加、删除节点。这两个功能在实际开发中非常重要，希望读者能够细细地体会以上代码的含义。

15.3.2 如何处理 TreeSelectionEvent 事件

当在 JTree 上點選任何一个节点时，都会触发 TreeSelectionEvent 事件，如果要处理该事件，必须实现 TreeSelectionListener 接口，此接口只定义了一个方法，即 valueChanged()方法，该方法主要是节点发生变化后触发的事件。TreeSelectionEvent 最常用于处理显示节点的内容，例如在文件图标中双击就可以看到文件的内容等。

在 JTree 中选择节点的方式共有 3 种，这 3 种情况与选择 JList 上的项目相同，如表 15.5 所示。

表 15.5 选择节点的方式

选择节点的方式	说明
DISCONTIGUOUS_TREE_SELECTION	可作单一选择，也可按住 Shift 键进行连续点的选择，或者按住 Ctrl 键，进行不连续多个节点的选择，这是 Java 默认值
CONTINUOUS_TREE_SELECTION	按住 Shift 键，可对某一连续的节点区间进行选取
SINGLE_TREE_SELECTION	一次只能选择一个节点

可以自行实现 TreeSelectionModel，用于制作更复杂的选择方式，但通常是没有必要的，因为 Java 提供了默认的选择模式类，即 DefaultTreeSelectionModel 类，利用该类可以很方便地设置上面 3 种选择模式。

下面列举一个有关树组件选择事件的实例，该实例主要用于创建一个有关类的树组件，通过单击树节点，在右边文本区中会显示出相应的内容。其具体的实例代码如下所示：

```
// 这段程序代码主要展示如何处理 TreeSelectionModel 事件
import java.awt.*;
import java.awt.event.ActionEvent;
```



```
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.*;

public class test6 implements TreeModelListener, TreeSelectionListener
{
    JLabel label;
    String nodeName = null; // 原有节点名称
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode node1;
    DefaultMutableTreeNode node2;
    DefaultMutableTreeNode node3;
    DefaultMutableTreeNode node4;
    JTree tree;
    DefaultTreeModel treeModel;
    JPanel p1;
    JPanel p2;
    JPanel p3;
    static JTextArea ta1;
    static JTextArea ta2;
    static JTextArea ta3;
    static JTextArea ta4;
    static JTextArea ta5;
    JPanel panel1;
    JPanel panel2;
    JPanel panel3;
    JPanel panel4;
    JPanel panel5;
    public test6()
    {
        p1=new JPanel();
        p2=new JPanel();
        p3=new JPanel();
        label = new JLabel("更改数据为:");
        ta1=new JTextArea(30,30);
        ta2=new JTextArea(40,40);
        ta3=new JTextArea(40,40);
        ta4=new JTextArea(40,40);
        ta5=new JTextArea(40,40);
        JTabbedPane tp=new JTabbedPane();
        panel1 = new JPanel ();
        panel2 = new JPanel ();
        panel3 = new JPanel ();
        panel4 = new JPanel ();
        panel5 = new JPanel ();
        tp.addTab("panel1", panel1);
        tp.setEnabledAt(0,true);
        tp.setTitleAt(0,"简介");
        tp.addTab ("panel2", panel2);
```



```

tp.setEnabledAt (1, true);
tp.setTitleAt (1, "用处");
tp.addTab ("panel3", panel3);
tp.setEnabledAt (2, true);
tp.setTitleAt (2, "各地比较");
tp.addTab ("panel4", panel4);
tp.setEnabledAt(0,true);
tp.setTitleAt(3,"一般进货来源");
tp.addTab ("panel5", panel5);
tp.setEnabledAt(4,true);
tp.setTitleAt(4," 备注");
tp.setPreferredSize (new Dimension (500,200));
tp.setTabPlacement (JTabbedPane.TOP);
tp.setTabLayoutPolicy (JTabbedPane.SCROLL_TAB_LAYOUT);
panel1.setLayout(new FlowLayout());
panel1.add(ta1);
panel2.setLayout(new FlowLayout());
panel2.add(ta2);
panel3.setLayout(new FlowLayout());
panel3.add(ta3);
panel4.setLayout(new FlowLayout());
panel4.add(ta4);
panel5.setLayout(new FlowLayout());
panel5.add(ta5);
JSplitPane splitPane = new JSplitPane ();
splitPane.setOneTouchExpandable (true);
splitPane.setContinuousLayout (true);
splitPane.setPreferredSize (new Dimension (100,200));
splitPane.setOrientation (JSplitPane.HORIZONTAL_SPLIT);
splitPane.setLeftComponent (p1);
splitPane.setRightComponent (p2);
splitPane.setDividerSize (3);
splitPane.setDividerLocation(50);
...

// 与上述内容中相同位置的代码相似, 这里不再列出
JSplitPane splitPane1 = new JSplitPane ();
splitPane1.setOneTouchExpandable (true);
splitPane1.setContinuousLayout (true);
splitPane1.setPreferredSize (new Dimension (100,200));
splitPane1.setOrientation (JSplitPane.VERTICAL_SPLIT);
splitPane1.setTopComponent (scrollPane);
splitPane1.setBottomComponent (label);
splitPane1.setDividerSize (1);
splitPane1.setDividerLocation(80);
p3.setLayout(new FlowLayout());
JSplitPane splitPane2 = new JSplitPane ();
splitPane2.setOneTouchExpandable (true);
splitPane2.setContinuousLayout (true);
splitPane2.setPreferredSize (new Dimension (100,200));
splitPane2.setOrientation (JSplitPane.VERTICAL_SPLIT);
splitPane2.setTopComponent (tp);
splitPane2.setBottomComponent (p3);
splitPane2.setDividerSize (1);
splitPane2.setDividerLocation(80);
p1.setLayout(new GridLayout(1,1));
p1.add(splitPane1);

```



```

p2.setLayout(new GridLayout(1,1));
p2.add(splitPane2);
f.setContentPane(splitPane);
f.pack();
f.setVisible(true);
tree.getSelectionModel().setSelectionMode(TreeSelectionMode.SINGLE_TREE_SELECTION);
// 设置树的选择模式
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
/* 本方法实现了 TreeModelListener 接口, 本接口共定义 4 个方法, 分别是 TreeNodesChanged()
* treeNodesInserted(), treeNodesRemoved(), treeNodesRemoved(),
* treeStructureChanged(), 在此范例中只针对更改节点值的部分, 因此只实现了
* treeNodesChanged()方法
*/
tree.addTreeSelectionListener(new TreeSelectionListener()
{ // 下面主要是相应树节点的选择事件
    public void valueChanged(TreeSelectionEvent e)
    {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) e.getPath().getLastPathComponent();
        // 将所选的节点模型化
        ta1.setText("You selected " + node);
    }
});
}
public void treeNodesChanged(TreeModelEvent e)
{ // 下面的方法是当节点发生改变时, 会在标签中显示出修改前的数据以及修改后的数据
    TreePath treePath = e.getTreePath();
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) treePath.getLastPathComponent();
    try
    {
        int[] index = e.getChildIndices();
        node = (DefaultMutableTreeNode) node.getChildAt(index[0]);
    }
    catch (NullPointerException exc) {}
    label.setText(nodeName+"更改数据为: "+(String)node.getUserObject());
}
public void treeNodesInserted(TreeModelEvent e)
{
}
public void treeNodesRemoved(TreeModelEvent e)
{
}
public void treeStructureChanged(TreeModelEvent e)
{
}
public void valueChanged(TreeSelectionEvent e)
{
    TreePath path = tree.getSelectionPath();
    if (path == null)
        return;
    DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode) path.getLastPathComponent();
    ta1.setText(selectedNode.toString());
}
}
public static void main(String args[])

```



```

    {
        new test6();
    }
}

```

上面程序代码的运行结果如图 15.11 所示。

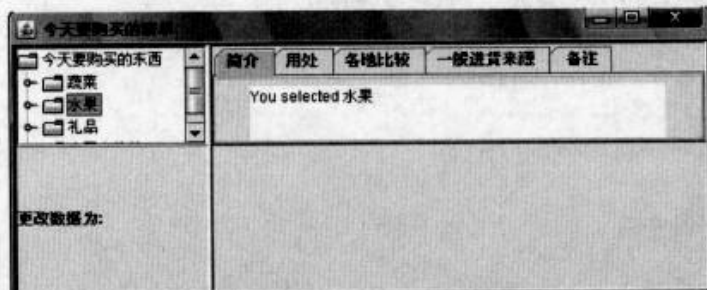


图 15.11 树组件选择事件

当在树组件中选择任意一个节点时，无论是枝节点还是叶节点，在右边的文本区中都会显示相应的信息，从而实现了树节点的选择事件。

下面将通过一个比较复杂的综合实例讲解树组件的创建以及树组件的事件处理方法。其程序代码如下所示：

```

// 这段程序代码主要展示如何处理 TreeSelectionModel 事件
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.*;

public class test7 implements TreeModelListener
{
    JLabel label;
    String nodeName = null; // 原有节点名称
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode node1;
    DefaultMutableTreeNode node2;
    DefaultMutableTreeNode node3;
    DefaultMutableTreeNode node4;
    JTree tree;
    DefaultTreeModel treeModel;
    JPanel p1;
    JPanel p2;
    JPanel p3;
    static JTextArea ta1;
    static JTextArea ta2;
    static JTextArea ta3;
    static JTextArea ta4;
    static JTextArea ta5;
    JPanel panel1;
}

```



```
JPanel panel2;  
JPanel panel3;  
JPanel panel4;  
JPanel panel5;  
public test7()  
{  
    ...  
    // 与上述内容中相同位置的代码相似, 这里不再列出  
    tree.addMouseListener(new MouseHandle());  
    class MouseHandle extends MouseAdapter  
    { // 处理 Mouse 点选事件  
        public void mousePressed(MouseEvent e)  
        {  
            String nodeName;  
            Try  
            { JTree tree = (JTree)e.getSource(); // 当单击鼠标, 在右边会显示出相应的数据信息  
              int rowLocation = tree.getRowForLocation(e.getX(), e.getY());  
              TreePath treepath = tree.getPathForRow(rowLocation);  
              TreeNode treenode = (TreeNode) treepath.getLastPathComponent();  
              nodeName = treenode.toString();  
              dd.ta1.setText("1 中含有丰富的碳水化合物、维生素和微量元素。  
                尤维生素 A 和胡萝卜素的含量较高。");  
              // 在右边文本区中显示的数据信息  
              dd.ta2.setText("可以吃、治病。 ");  
              dd.ta3.setText("上海的比较便宜。 ");  
              dd.ta4.setText("一般进货从广东惠州。 ");  
              dd.ta5.setText("营养价值高, 需要多买点。 ");  
            }  
            catch(NullPointerException ne){}  
            ...  
        }  
    }  
}
```

上面程序代码的运行结果如图 15.12 所示。

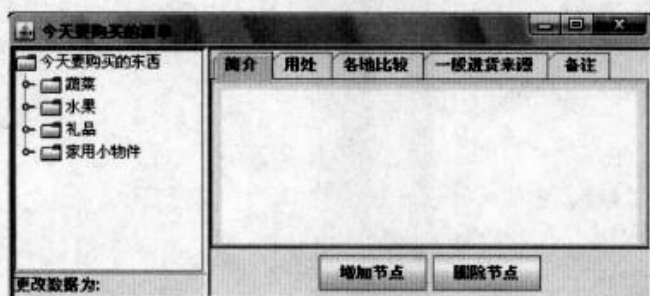


图 15.12 综合实例 (a)

当单击任意一个节点时, 会弹出如图 15.13 所示的效果图。

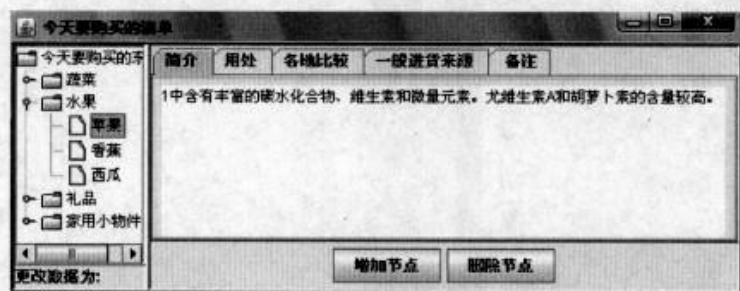


图 15.13 综合实例 (b)

15.4 如何定义个性化树

本节将为读者讲述一些树组件的显示特性，它们能够使开发人员创建出具有个性化的树组件。下面将常用的一些显示特性列出，如表 15.6 所示。

表 15.6 树组件的显示特性

方法	说明
<code>void setRootVisible(boolean rootVisible)</code>	可以设置是否隐藏根节点
<code>Void setShowsRootHandles(boolean newValue)</code>	设置是否显示节点前面的加号
<code>void putClientProperty(Object key,Object value)</code>	设置节点之间的连线样式
<code>void setClosedIcon(Icon icon)</code>	设置非展开时的图标
<code>void setOpenIcon(Icon newIcon)</code>	设置节点展开时的图标
<code>void setLeafIcon(Icon newIcon)</code>	设置叶节点的图标

将以上方法按照自己的意愿放入程序中，则所设计的树组件将会很具个性化。在实际开发中，还会涉及到渲染器方面的知识，它的用法和 `JTable` 的渲染器相似。限于篇幅问题，这里无法给予详细说明和举例，有兴趣的读者可以自行查阅资料。

15.5 树组件的常用 API

本节将为读者总结有关树组件的常用 API，如表 15.7 所示。

表 15.7 与树相关的类和接口的 API

类或接口	说明
<code>JTree</code>	代表树组件
<code>TreePath</code>	代表节点的路径
<code>TreeNode</code>	这三个都是接口类型，要求树节点必须实现的
<code>MutableTreeNode</code>	
<code>DefaultMutableTreeNode</code>	
<code>TreeModel</code>	这两个是接口类型，是创建树模型必须实现的
<code>DefaultTreeModel</code>	
<code>TreeCellRenderer</code>	表示树单元格的渲染器
<code>DefaultTreeCellRenderer</code>	
<code>TreeCellEditor</code>	表示树单元格的编辑器
<code>DefaultTreeCellEditor</code>	
<code>TreeSelectionModel</code>	树组件的选择事件模型
<code>DefaultTreeSelectionModel</code>	
<code>TreeSelectionListener</code>	树组件的选择事件的侦听器
<code>DefaultTreeSelectionListener</code>	
<code>TreeModelListener</code>	树组件模型事件侦听器
<code>TreeModelEvent</code>	

创建和设置树的 API 的说明如表 15.8 所示。

表 15.8 创建和设置树的 API

构造器或方法	说明
JTree(TreeNode)	创建树组件的构造器
JTree(TreeNode,Boolean)	
JTree(TreeModel)	
JTree()	
JTree(Hashtable)	
JTree(Object[])	
JTree(Vector)	
Void setCellRenderer(TreeCellRenderer)	绘制一个节点的渲染器
Void setEditable(Boolean)	设置树节点是否可以编辑
Void setCellEditor(TreeCellEditor)	设置自定义的节点编辑器
Void setRootVisible(Boolean)	设置是否应该显示根节点
Void setShowRootHandles(Boolean)	设置是否显示根节点展开或折叠时的操作柄
Void setDragEnabled(Boolean)	设置拖动属性
Boolean getDragEnabled()	获得拖动属性

上面给出了常用的 API，希望读者能够参照上面的表格，多多举例练习，这样才能真正掌握树组件在实际开发中的使用方法。

15.6 本章小结

本章主要讲述了有关树组件的相关知识，如创建树组件的三种方法、如何响应树组件的事件等。其实，在实际开发中还有很多相关的知识，在这里不可能全部给予讲述，希望读者能够以本章作为基础，进一步深究其中的奥秘。

15.7 本章习题

1. 使用哈希表创建一个树组件，这个树组件是关于家庭用品的，包括家用电器、厨具、家具、书。

要求：其最终结果如图 15.14 所示。



图 15.14 本章习题 1

2. 使用树模型创建上例中的树组件。
3. 针对本章习题 1 增加一些功能，当单击某个节点时，会在下面的标签中显示出此节点。
要求：
(1) 其最终结果如图 15.15 所示。

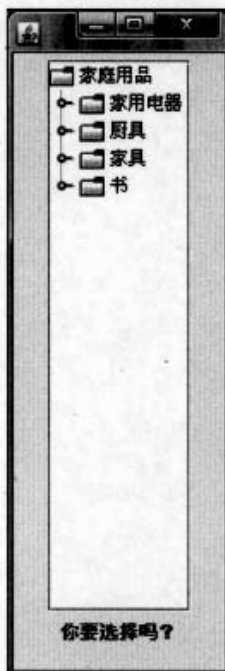


图 15.15 本章习题 3 (a)

- (2) 当选择任意一项后，会在标签组件中显示出来，如图 15.16 所示。

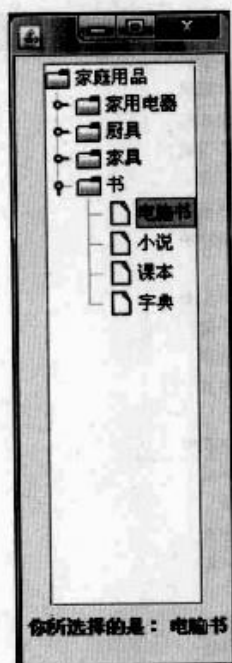


图 15.16 本章习题 3 (b)

4. 针对本章习题 1 增加一些功能，当单击某个节点时，会在一旁的文本组件中显示出相关的内容。

要求：其最终结果与上例相同。

第16章 如何使用Swing观感器

本章主要介绍有关 Swing 观感器方面的知识，所谓观感器也就是创建 Swing 开发的程序界面的 API。一个软件程序界面的风格直接体现了开发人员的专业性与否，而且一个软件程序界面的交互界面的风格好坏对用户的操作起着非常重要的作用，例如有的用户习惯于使用类似 Linux 的界面，有的用户则习惯于使用类似 XP 的界面，于是可以通过 Swing 观感器来设计适合和符合用户口味的软件程序界面，从而让用户能够得心应手地操作程序。

16.1 如何设置程序的观感

Swing UI 管理器是对 Swing 组件的外观和显示进行控制。在运行一个图形化程序时，Swing UI 管理器必须确定程序使用的是哪一种外观感觉，如果软件本身没有定义外观感觉，那么 Swing UI 管理器将选择默认的 Java 外观感觉。如果软件本身已经设置了外观感觉，那么 Swing UI 管理器将选择软件本身所设置的外观感觉。

Swing 使用一种可插式外观和感觉（Pluggable Look-And-Feel, PLAF）体系结构。开发人员无须针对不同的组件颜色和字体之类的设置进行硬编码。组件直接从 Swing UI 管理器请求这些设置。开发人员通过 UI 管理器选择不同的界面，其中有不同的界面风格，如可以选择 Windows、Motif 或 Metal 样式等。UI 管理器还可以设置每个组件应该如何显示它自身。

设置程序的观感分为两步：第一步是通过编程的方式设置外观感觉，第二步就是指定外观感觉。下面将分别讲述它们的方法。

在这里有一点需要强调：设置观感的语句应该放在应用程序的最开始位置，因为如果不是这样的话，很有可能会被默认的 Java 观感器所覆盖掉。

如果通过编程指定一个外观感觉的话，一般可使用方法 `UIManager.setLookAndFeel()`，括号内为指定的外观感觉。如果要指定 Java 外观感觉，可以使用方法 `getCrossPlatformLookAndFeelClassName()`，如果要指定当前运行程序所在平台的本地外观感觉，可以使用方法 `getSystemLookAndFeelClassName`，如果要指定一个特定的观感，那么就必须输入其确切的名称。下面将列举出一些可以在 `setLookAndFeel` 方法中使用的参数，如表 16.1 所示。

表 16.1 setLookAndFeel 方法中使用的参数

方法	说明
UIManager.getCrossPlatformLookAndFeelClassName()	返回适用于所有平台的观感
UIManager.getSystemLookAndFeelClassName()	指定当前平台的观感
Com.sun.java.plaf.gtk.GTKLookAndFeel	指定 GTK+ 观感
Com.sun.java.swing.plaf.windows.windowsLookAndFeel	指定窗口外观感觉, 但只能在 Windows 系统中适用
Javax.swing.plaf.metal.MetalLookAndFeel	指定了名称为 Metal 的 Java 观感
Com.sun.java.swing.plaf.motif.MotifLookAndFeel	指定 CDE/Motif 的观感

对于 Java 开发出来的程序, 当然希望其程序界面能够在不同的系统中仍能保持原样, 开发人员可通过 UIManager.getCrossPlatformLookAndFeelClassName() 方法构建一个组件, 这种组件具有一定样式和行为, 无论在哪种操作系统中运行都能够保持不变风格的界面, 它是不依赖于操作系统平台的可插入外观, 使得用户无须重启就可以切换外观。

Swing 的可插入外观的 API 可以支持以下几种特性:

- 默认外观, 也被称作 Java 外观, 以前被称为 Metal 外观。它可以跨平台保持 Java 外观。
- 系统外观, 它是基于地层操作系统的外观。
- 自定义外观, 这是由程序开发人员自行定义的。

下面将给出一个简单的实例, 使读者能够充分了解观感器的使用方法。其具体的代码如下所示:

```
// 这段程序代码主要是为读者展示使用观感器的图形界面与不使用观感器的图形界面的比较
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;

public class test1
{
    public test1()
    {
        JFrame f=new JFrame("学校名录");
        Container contentPane=f.getContentPane();
        DefaultMutableTreeNode root=new DefaultMutableTreeNode("学校名录");
        // 使用 DefaultMutableTreeNode 的构造器创建根节点
        DefaultMutableTreeNode node1=new DefaultMutableTreeNode("教导处");
        // 使用 DefaultMutableTreeNode 的构造器创建 4 个枝节点
        DefaultMutableTreeNode node2=new DefaultMutableTreeNode("一年级");
        DefaultMutableTreeNode node3=new DefaultMutableTreeNode("二年级");
        DefaultMutableTreeNode node4=new DefaultMutableTreeNode("三年级");
        root.add(node1); // 将 4 个枝节点添加到根节点中
        root.add(node2);
        root.add(node3);
        root.add(node4);
        DefaultMutableTreeNode leafnode=new DefaultMutableTreeNode("王成");
        // 利用 DefaultMutableTreeNode 的构造器创建出叶节点, 再将叶节点分别添加到不同的枝节点上
        node1.add(leafnode);
        leafnode=new DefaultMutableTreeNode("赵薇");
        node1.add(leafnode);
    }
}
```



```
leafnode=new DefaultMutableTreeNode("李明");
node1.add(leafnode);
leafnode=new DefaultMutableTreeNode("宋大兵");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("雷宝");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("赵月");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("潘良");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("严康");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("王鹏");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("刘华");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("朱卫");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("谭豪");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("叶婷");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("陈浩");
node4.add(leafnode);
JTree tree=new JTree(root);
JScrollPane scrollPane=new JScrollPane();
scrollPane.setViewportView(tree);
contentPane.add(scrollPane);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String[] args)
{
    try
    {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e){}
    new test1();
}
```

下面将比较添加观感器和没有观感器的运行结果。没有添加观感器的程序代码的运行结果如图 16.1 所示。



图 16.1 没有添加观感器

添加了观感器的程序代码的运行结果如图 16.2 所示。

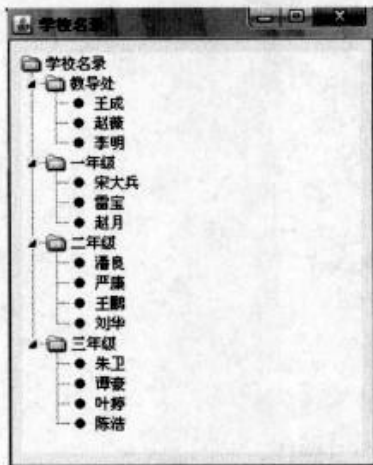


图 16.2 添加了观感器

添加了观感器的程序界面是否比没有添加观感器的程序界面更加熟悉、更加友好呢。上面的实例只是针对 `UIManager.getSystemLookAndFeelClassName()` 观感器给予展示。其他的观感器读者可以自行尝试。

在实际开发中，将特殊的观感器插件下载后（此插件是一个“.jar”文件），将此文件放置到“Java\jre1.6.0_03\lib\ext”文件夹中，于是就可以在编辑器中使用它了。

16.2 如何自定义观感器

本节主要讲述如何自定义观感器，自定义观感器可以让开发人员根据用户的特殊要求进行特殊的开发，例如用户需要一个图形界面中某些组件表现为一种风格的观感器，而其他某些组件需要表现为另一种风格的观感器，遇到此种情况应该如何处理呢？本节将进行详细介绍。

为了能够让读者熟悉上面所列出的情况，下面给出一个实例，请读者参照实例思考如何才能创建自己的外观子类。其程序代码如下所示：

```
// 这段程序代码主要为读者展示如何为每个组件定义一个观感器
import java.awt.*;
```



```

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;
import javax.swing.tree.*;

public class test2
{
    static final int WIDTH=700;
    static final int HEIGHT=400;
    JLabel label;
    String nodeName = null; // 原有节点名称
    JPanel p1;
    JPanel p2;
    JPanel p3;
    public test2()
    {
        JFrame f=new JFrame("今天要购买的清单");
        p1=new JPanel();
        p2=new JPanel();
        p3=new JPanel();
        label = new JLabel("更改数据为: ");
        JSplitPane splitPane = new JSplitPane ();
        splitPane.setOneTouchExpandable (true);
        splitPane.setContinuousLayout (true);
        splitPane.setPreferredSize (new Dimension (100,200));
        splitPane.setOrientation (JSplitPane.HORIZONTAL_SPLIT);
        splitPane.setLeftComponent (p1);
        splitPane.setRightComponent (p2);
        splitPane.setDividerSize (3);
        splitPane.setDividerLocation(50);
        lookandfeel1 lf=new lookandfeel1(); // 创建 lookandfeel1 对象
        JScrollPane scrollPane = new JScrollPane(); // 将树组件添加到滚动条组件中
        scrollPane.setViewPortView(lf.tree);
        JSplitPane splitPane1 = new JSplitPane ();
        splitPane1.setOneTouchExpandable (true);
        splitPane1.setContinuousLayout (true);
        splitPane1.setPreferredSize (new Dimension (100,200));
        splitPane1.setOrientation (JSplitPane.VERTICAL_SPLIT);
        splitPane1.setTopComponent (scrollPane);
        splitPane1.setBottomComponent (label);
        splitPane1.setDividerSize (1);
        splitPane1.setDividerLocation(80);
        JButton b1=new JButton("增加节点");
        JButton b2=new JButton("删除节点");
        p3.setLayout(new FlowLayout());
        p3.add(b1);
        p3.add(b2);
        JSplitPane splitPane2 = new JSplitPane ();
        splitPane2.setOneTouchExpandable (true);
        splitPane2.setContinuousLayout (true);
        splitPane2.setPreferredSize (new Dimension (100,200));
        splitPane2.setOrientation (JSplitPane.VERTICAL_SPLIT);
        lookandfeel2 lf2=new lookandfeel2(); // 创建 lookandfeel2 对象
        splitPane2.setTopComponent (lf2.tp); // 将 JTabbedPane 组件添加到面板中
        splitPane2.setBottomComponent (p3);
        splitPane2.setDividerSize (1);
        splitPane2.setDividerLocation(80);
    }
}

```



```

p1.setLayout(new GridLayout(1,1));
p1.add(splitPane1);
p2.setLayout(new GridLayout(1,1));
p2.add(splitPane2);
f.setContentPane(splitPane);
f.setSize(WIDTH,HEIGHT);
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);
f.pack();
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
public static void main(String args[])
{
    new test2();
}
}
class lookandfeel1
{
    /* 创建一个 lookandfeel1 类, 这个类主要用于创建一个树组件
    * 之所以放到单独一个类的原因是要为这个组件单独规划一个观感器
    */
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode node1;
    DefaultMutableTreeNode node2;
    DefaultMutableTreeNode node3;
    DefaultMutableTreeNode node4;
    static JTree tree;
    DefaultTreeModel treeModel;
    lookandfeel1()
    {
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e){}
        root=new DefaultMutableTreeNode("今天要购买的东西");
        node1=new DefaultMutableTreeNode("蔬菜");
        node2=new DefaultMutableTreeNode("水果");
        node3=new DefaultMutableTreeNode("礼品");
        node4=new DefaultMutableTreeNode("家用小物件");
        root.add(node1);
        root.add(node2);
        root.add(node3);
        root.add(node4);
        DefaultMutableTreeNode leafnode=new DefaultMutableTreeNode("白菜");
    }
}

```



```

node1.add(leafnode);
leafnode=new DefaultMutableTreeNode("大蒜");
node1.add(leafnode);
leafnode=new DefaultMutableTreeNode("土豆");
node1.add(leafnode);
leafnode=new DefaultMutableTreeNode("苹果");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("香蕉");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("西瓜");
node2.add(leafnode);
leafnode=new DefaultMutableTreeNode("礼品");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("茅台酒");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("营养麦片");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("保健食品");
node3.add(leafnode);
leafnode=new DefaultMutableTreeNode("味精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("酱油");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("洗洁精");
node4.add(leafnode);
leafnode=new DefaultMutableTreeNode("保险袋");
node4.add(leafnode);

tree=new JTree(root);
tree.setEditable(true);// 设置 JTree 为可编辑的
// 下面两行取得 DefaultTreeModel, 并检测是否有 TreeModelEvent 事件
final DefaultTreeModel treeModel = (DefaultTreeModel)tree.getModel();
}
}
class lookandfeel2
{
    /*创建一个 lookandfeel2 类, 这个类主要是创建一个树组件
    * 之所以放到单独一个类的原因是要为这个组件单独规划一个观感器
    */
    JPanel panel1;
    JPanel panel2;
    JPanel panel3;
    JPanel panel4;
    JPanel panel5;
    static JTextArea ta1;
    static JTextArea ta2;
    static JTextArea ta3;
    static JTextArea ta4;
    static JTextArea ta5;
    static JTabbedPane tp;
    lookandfeel2()
    {
        try
        {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        }
        catch(Exception e){}
    }
}

```



```
ta1=new JTextArea(30,30);
ta2=new JTextArea(40,40);
ta3=new JTextArea(40,40);
ta4=new JTextArea(40,40);
ta5=new JTextArea(40,40);
tp=new JTabbedPane();
panel1 = new JPanel ();
panel2 = new JPanel ();
panel3 = new JPanel ();
panel4 = new JPanel ();
panel5 = new JPanel ();
tp.addTab("panel1", panel1);
tp.setEnabledAt(0,true);
tp.setTitleAt(0,"简介");
tp.addTab("panel2", panel2);
tp.setEnabledAt(1, true);
tp.setTitleAt(1,"用处");
tp.addTab("panel3", panel3);
tp.setEnabledAt(2, true);
tp.setTitleAt(2,"各地比较");
tp.addTab("panel4", panel4);
tp.setEnabledAt(0,true);
tp.setTitleAt(3,"一般进货来源");
tp.addTab("panel5", panel5);
tp.setEnabledAt(4,true);
tp.setTitleAt(4,"备注");
tp.setPreferredSize(new Dimension(500,200));
tp.setTabPlacement(JTabbedPane.TOP);
tp.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
panel1.setLayout(new FlowLayout());
panel1.add(ta1);
panel2.setLayout(new FlowLayout());
panel2.add(ta2);
panel3.setLayout(new FlowLayout());
panel3.add(ta3);
panel4.setLayout(new FlowLayout());
panel4.add(ta4);
panel5.setLayout(new FlowLayout());
panel5.add(ta5);
}
```

上面程序代码的运行结果如图 16.3 所示。

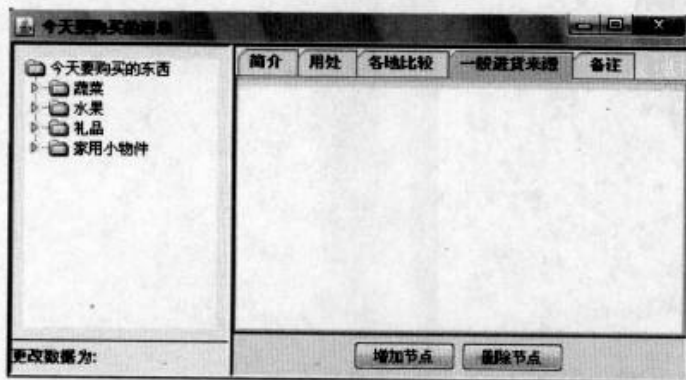


图 16.3 设计自己的观感器

以上程序实例主要展示了在一个程序界面中两种不同风格的观感器，左窗格的树组件是使用 Windows 风格的观感器，而右窗格则是依然沿用 Java 默认的风格观感器。

在实际开发中，开发人员可以根据以上方法，在相同的界面上、不同的组件中，设置成不同风格的观感器，使得用户界面变得更加个性化和专业化。

16.3 本章小结

本章主要讲述了如何创建 Swing 观感器，并通过实例讲解如何自定义观感器。在自定义观感器一节中，总结了在实际开发中如何灵活的运用所讲述的方法进行个性化的设置和开发，从而能够满足用户对软件界面外观的各种需求，进一步体现了开发人员对 Java Swing 图形化开发的熟悉程度，其实在有些公司的面试中，也会涉及到此类题目。

16.4 本章习题

1. 什么是观感器？
2. Swing 的可插入外观的 API 可以支持哪些特性？
3. 如何学会定义自己的观感？
4. 在实际开发中，如何使用一些特殊的观感器插件？
5. 设计一个登录窗口程序，让其中的各个组件所显示的效果不同。

要求：其最终结果如图 16.4 所示。



图 16.4 本章习题 5

第17章 Swing与并发

假定读者已经具备了 Java 线程的基础知识，所以不再叙述有关线程方面的概念了。本章将花费大的篇幅介绍有关 Swing 多线程与并发的原理以及使用方面的知识。其中包括了不同种类的线程的概念和作用、背景任务、临时结果任务等。而其中的多线程种类则包括了启动线程、事件分派线程、工作线程。在介绍理论的同时，还配上丰富的实例，让读者能够充分地理解每一个概念以及其使用方法，从而为以后实际开发工作奠定良好的基础。

17.1 多线程问题

为什么 Swing 中会出现多线程的问题呢？现在假设一种情况：有一个使用 Swing 开发出来的软件界面，它是采用顺序编程的，也就是说程序代码是依次执行的，这种情况会对客户造成一个很严重的影响。这个影响就是当一个客户要执行软件界面中的一个功能时，用户界面就会停止在那里，此时用户什么也干不了，只能等待。

例如软件界面上有一个下载功能，那么用户只能静静的等待下载完毕，才能继续其他的操作。如果说这个下载任务时间很短则无关紧要，如果需要等待的时间很长的话，相信这样的软件，用户一定是无法接受的。

那么为了解决这个问题，Swing 的多线程出现了，它会将一些比较耗时的任务单独放到一个独立的线程中进行处理，用户可以使用线程继续其他的操作。这样的话，软件就可以将本身的功效发挥到极致。

Swing 多线程中有三种不同的线程。

- 初始线程：初始线程主要干两件事情，一是创建一个可运行的对象，这个可运行对象主要负责初始化图形界面，二是将这个可运行的对象安排到事件分派线程中去执行。
- 事件分派线程：事件分派线程主要是用来处理 Swing 中的方法，因为大部分 Swing 对象中的方法都不是线程安全的，只有少数是线程安全的，所以需要事件分派线程来保证线程安全。
- 工作线程：一般用来处理一些耗时的任务，例如一些下载的程序或者一些需要等待很长时间的程序。

上面的简单介绍使读者有了一个初步的概念，下面将会针对以上三种不同种类的线程，给予详细说明。

17.2 初始线程

初始线程主要用来创建 GUI 组件、资源加载和启动被创建的 GUI 组件。当 GUI 组件出现后，初始线程的任务就基本完成了，接下来它利用 `SwingUtilities` 的 `invokeLater()` 和 `invokeAndWait()` 方法来实现的 Swing 事件处理交给事件分派线程。本节主要讲述这两个方法的含义和用法。下面看一下初始线程的一般性代码是如何编写的，其代码如下所示：

```
SwingUtilities.invokeLater(new Runnable()
{
    public void run()
    {
        方法体;
    }
});
```

下面将分析上面程序代码的含义。

- `SwingUtilities` 类是一个 Swing 实用方法集合类。
- `invokeLater(Runnable doRun)` 方法是让 `doRun.run()` 在 Swing 事件指派线程上异步执行。在编写代码时，必须把要运行的代码放到一个 `Runnable` 对象的 `run()` 方法中，并将此 `Runnable` 对象设为 `invokeLater()` 的参数。`invokeLater()` 方法接受到参数后会立即返回，而无须等待事件派发线程执行指定代码，就可以继续执行下面的代码。
- `invokeAndWait(Runnable doRun)` 方法是让 `doRun.run()` 在 Swing 事件指派线程上同步执行。此方法是等事件派发线程执行了指定代码后才返回执行自己下面的代码，它相当于一个函数调用过程。
- 方法体就是一个 `Runnable` 类型的数据。

为了能够充分理解 `SwingUtilities` 的 `invokeLater()` 和 `invokeAndWait()` 方法的使用，下面将通过一些程序代码片段来讲述它们。

先来看一个有关 `invokeLater()` 方法的实例，其实例代码如下所示：

```
Runnable ExampleRunnable = new Runnable()
{
    public void run()
    {
        example();
    }
};
SwingUtilities.invokeLater(ExampleRunnable);
```

这个程序代码的意义就是将 `ExampleRunnable` 对象中的 `run()` 方法送至事件分派线程中执行，然后立即返回，无须等待执行 `run()` 方法中的方法体，就直接执行接下来的其他代码。

再来看一个有关 `invokeAndWait()` 方法的实例，其实例代码如下所示：

```
Runnable ExampleRunnable = new Runnable()
{
    public void run()
    {
        example();
    }
};
```



```
};  
SwingUtilities.invokeLater(ExampleRunnable);
```

以上程序代码的意义是将 `ExampleRunnable` 对象中的 `run()` 方法送至事件分派线程中执行, 然后等待执行 `run()` 方法中的方法体完成后, 再开始执行接下来的其他代码。

下面总结一下这两个方法。

- `invokeAndWait`: 后面的程序必须等这个线程 (参数中的线程) 执行完才能执行。
- `invokeLater`: 后面的程序和这个参数的线程对象可以并行或者异步地执行。
`invokeLater` 一般用于在线程里修改 Swing 组件的外观, 因为 Swing 组件是非同步的, 所以不能在线程中直接修改, 会不同步, 得不到期望的效果, 所以要把修改外观的代码放在一个单独的线程中, 交给 `invokeLater`。

通过上面的讲述, 读者应该可以理解初始线程的概念, 以及它是如何将事件处理代码交给事件分派线程处理的。

下面继续为读者讲解第二种线程——事件分派线程, 它也是一个非常重要和关键的线程, 因为它需要处理 Swing 程序中绝大多数的代码。

17.3 事件分派线程

本节将结合实例为读者深入讲述事件分派线程以及如何使用。

Swing 事件处理线程和绘制代码是在单个线程中执行的。该线程被称为事件分派线程。这样做可以保证每一个事件处理器执行代码时是封闭的, 也就是说, 每一个事件处理器在处理下一个事件之前不会被终止, 从而保证了程序代码被正常执行。其实, 和这个线程紧密相关的是一个 FIFO 事件的队列——系统事件队列, 它是 `java.awt.EventQueue` 的一个实例, 每个事件都是按顺序一个一个执行的。下面将先给出一个实例, 通过这个实例, 让读者能够深刻地了解这种线程的工作方式。其具体代码如下所示:

```
// 这段程序代码主要为读者展示如何使用 SwingUtilities.invokeLater()方法来处理不同的线程  
import javax.swing.*;  
class process extends Thread  
{  
    static final int WIDTH=700;  
    static final int HEIGHT=400;  
    public process()  
    {  
        JFrame frame=new JFrame("Swing 多线程测试程序");  
        JPanel panel=new JPanel();  
        frame.setContentPane(panel);  
        JButton button1=new JButton("按钮一");  
        JButton button2=new JButton("按钮二");  
        JButton button3=new JButton("按钮三");  
        JButton button4=new JButton("按钮四");  
        panel.setLayout(new GridLayout(2,2));  
        panel.add(button1);  
        panel.add(button2);  
        panel.add(button3);  
        panel.add(button4);  
        Toolkit kit=Toolkit.getDefaultToolkit();  
        Dimension screenSize=kit.getScreenSize();
```



```

        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setVisible(true);
        frame.setResizable(false);
    }
}

public class test1
{
    static process p;
    public static void main(String[] args)
    {
        try
        {
            SwingUtilities.invokeLater(new Runnable()
            {
                public void run()
                {
                    p=new process();
                    try
                    {
                        p.sleep(30000);
                    }
                    catch(Exception e){}
                }
            });
        }
        catch(Exception e){}
        JFrame frame=new JFrame("Swing 多线程测试程序");
        JPanel panel=new JPanel();
        frame.setContentPane(panel);
        JButton button1=new JButton("按钮一");
        JButton button2=new JButton("按钮二");
        JButton button3=new JButton("按钮三");
        JButton button4=new JButton("按钮四");
        panel.setLayout(new GridLayout(2,2));
        panel.add(button1);
        panel.add(button2);
        panel.add(button3);
        panel.add(button4);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-process.WIDTH)/2;
        int y=(height- process.HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setVisible(true);
        frame.setResizable(false);
    }
}

```

运行的结果会出现两个窗口界面，并且是同时出现的，那为什么会出现两个界面呢？因为 `SwingUtilities.invokeLater()` 把所要执行的线程交付给事件分派线程后，就直接开始执行下

面的程序段了，所以会同时出现两个界面。如果将上面的程序中的 `SwingUtilities.invokeLater()` 改成 `SwingUtilities.invokeAndWait()` 的话，等待 30 秒后，第二个窗口才会出现，从而验证了前面所说的两个方法的执行方式不同。

在即将结束讲述事件分派线程之前，需要提醒读者：为了避免线程的执行，所以尽量把创建组件的代码通过 `SwingUtilities.invokeLater()` 方法放入到事件分派线程中执行，不要单独放到 `main` 程序中创建组件，因为这样很有可能会出现事件处理和创建组件的先后顺序混乱的情况，从而导致线程死锁。

17.4 工作线程

由本章的第一节可以知道，工作线程主要是运行一些比较耗时的程序，而本节将详细讨论有关工作线程和 `SwingWorker` 的知识。

当一个 `Swing` 程序需要执行一个比较耗时的任务时，一般是通过使用一个工作线程来完成的。一个任务在一个工作线程中被执行。在 `JDK 1.6` 之前，一般工作线程是由程序开发人员自己去定义的，而自从 `JDK 1.6` 发布后，工作线程则完全由一个被称作 `SwingWorker` 的类来处理。

那什么是 `SwingWorker` 类呢？`SwingWorker` 类是一个抽象类，所以，要使用它就必须通过定义它的子类来创建 `SwingWorker` 的对象。`SwingWorker` 的子类可以定义方法，当背景任务完成后，它将自动被事件分派线程调用。同时，`SwingWorker` 类实现了 `java.util.concurrent.Future` 接口，这个接口允许背景任务提供一个返回值给其他线程，也可以允许撤销背景任务以及确定背景任务究竟是被完成了还是被撤销了。

下面将详细讲述有关如何使用 `SwingWorker` 类处理工作线程的知识，因为只要掌握了如何使用 `SwingWorker` 类，也就掌握了如何使用工作线程运行。

要使用 `SwingWorker` 类，首先必须创建一个子类，需要强调的是这个子类必须要实现一个被称作 `doInBackground()` 的方法，在这个方法内的方法体就是耗时的任务。其实，`doInBackground()` 方法相当于一个耗时线程的构造器。当初始化 `SwingWorker` 子类时，它将会创建一个线程，并且要通过 `execute()` 方法来启动这个线程，同时这个方法也就调用了 `doInBackground()` 方法。

这就是 `SwingWorker` 的一个工作步骤，希望读者能够充分地理解它，因为这个步骤就是最简单的工作线程处理过程。下面将给出一个程序片段使读者有个概念性的认识，其程序代码片段如下所示：

```
public void actionPerformed(ActionEvent e)
{
    SwingWorker worker=new SwingWorker()
    {
        protected Object doInBackground() throws Exception
        {
            // 耗时任务的代码
            return 某些值;
        }
    };
    worker.execute();
}
```


这样是否觉得非常简单呢？的确，只要掌握了如何利用 `SwingWorker` 类处理工作线程的原理和步骤，那么就很容易理解上面的程序代码片段。

17.4.1 简单的背景任务

所谓的背景任务也就是指使用工作线程来完成的任务，本小节将给出一个实例，通过实例让读者能够真正地掌握如何利用工作线程来处理背景任务。这个程序创建了一个主窗口，在主窗口中有 6 个按钮，每个按钮的动作事件都会打开一个窗口，本程序需要测试的是在执行一个按钮动作事件时，是否可以再执行另一个按钮动作事件，如果行的话说明这些动作事件都被工作线程转入后台执行了。其具体的程序代码如下所示。

```
// 这段代码主要是为读者展示如何处理背景任务，在程序中有 6 个顶层容器
// 可以在同一时间内打开，因为它们都处于工作线程
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class test2
{
    public static void main(String[] args)
    {
        try
        {
            SwingUtilities.invokeLater(new Runnable()
            {
                public void run()
                {
                    new mainframe();
                }
            });
        }
        catch(Exception e){}
    }
}
```

下面将给出一个继承了线程类的类，这个线程类主要是控制 6 个按钮组件，每个按钮组件作为线程，它们之间相隔一段时间，其具体代码如下所示。

```
class mainframe extends Thread
{
    static final int WIDTH=700;
    static final int HEIGHT=400;
    mainframe()
    {
        JFrame frame=new JFrame("SwingWorker 测试程序");
        JPanel pane=new JPanel();
        frame.setContentPane(pane);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
```



```

frame.setLocation(x,y);
frame.setVisible(true);
frame.setResizable(true);
JButton button1=new JButton("打开第一个窗口");
JButton button2=new JButton("打开第二个窗口");
JButton button3=new JButton("打开第三个窗口");
JButton button4=new JButton("打开第四个窗口");
JButton button5=new JButton("打开第五个窗口");
JButton button6=new JButton("打开第六个窗口");
pane.setLayout(new GridLayout(2,3));
pane.add(button1);
pane.add(button2);
pane.add(button3);
pane.add(button4);
pane.add(button5);
pane.add(button6);
button1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        SwingWorker work=new SwingWorker()
        {
            protected Object doInBackground() throws Exception
            {
                try
                {
                    sleep(10000);
                }
                catch(Exception e){}
                frame1 f=new frame1();
                return null;
            }
        };
        work.execute();
    }
});
button2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        SwingWorker work=new SwingWorker()
        {
            protected Object doInBackground() throws Exception
            {
                try
                {
                    sleep(10000);
                }
                catch(Exception e){}
                frame2 f=new frame2();
                return null;
            }
        };
        work.execute();
    }
});

```



```
button3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        SwingWorker work=new SwingWorker()
        {
            protected Object doInBackground() throws Exception
            {
                try
                {
                    sleep(10000);
                }
                catch(Exception e){}
                frame3 f=new frame3();
                return null;
            }
        };
        work.execute();
    }
});
button4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        SwingWorker work=new SwingWorker()
        {
            protected Object doInBackground() throws Exception
            {
                try
                {
                    sleep(10000);
                }
                catch(Exception e){}
                frame4 f=new frame4();
                return null;
            }
        };
        work.execute();
    }
});
button5.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        SwingWorker work=new SwingWorker()
        {
            protected Object doInBackground() throws Exception
            {
                try
                {
                    sleep(10000);
                }
                catch(Exception e){}
                frame5 f=new frame5();
                return null;
            }
        }
    }
});
```



```

        };
        work.execute();
    }
});
button6.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        SwingWorker work=new SwingWorker()
        {
            protected Object doInBackground() throws Exception
            {
                try
                {
                    sleep(10000);
                }
                catch(Exception e){}
                frame6 f=new frame6();
                return null;
            }
        };
        work.execute();
    }
});
}
...
}

```

下面将给出第 1 个按钮打开的第 1 个窗口创建的代码：

```

class frame1
{ // 创建一个窗口
    frame1()
    {
        JFrame frame=new JFrame("SwingWorker 测试程序 1");
        JPanel pane=new JPanel();
        frame.setContentPane(pane);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-mainframe.WIDTH)/2;
        int y=(height-mainframe.HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setVisible(true);
        frame.setResizable(false);
        JButton button1=new JButton("这是第一个窗口的结果");
        pane.add(button1,"Center");
    }
}

```

下面将给出第 2~6 个按钮组件打开的第 2~6 个窗口的具体代码：

```

class frame2
{ // 创建一个窗口
    frame2()
    {

```



```
...
// 与第 1 个窗口中的代码内容相同
}
}
class frame3
{ // 创建一个窗口
    frame3()
    {
        ...
        // 与第 1 个窗口中的代码内容相同
    }
}
class frame4
{ // 创建一个窗口
    frame4()
    {
        ...
        // 与第 1 个窗口中的代码内容相同
    }
}
class frame5
{ // 创建一个窗口
    frame5()
    {
        ...
        // 与第 1 个窗口中的代码内容相同
    }
}
class frame6
{
    frame6()
    {
        ...
        // 与第 1 个窗口中的代码内容相同
    }
}
```

上面的程序代码执行后，会弹出如图 17.1 所示界面。



图 17.1 背景任务

以上程序可以同时单击 6 个按钮控件，打开 6 个窗口，并且程序不会发生停滞。因为已经把所有的鼠标单击事件放到了后台去运行，不会造成前台程序的运行。在程序代码中，使用了 `sleep()` 方法的目的是为了让程序执行的时间长一些，从而能够观察到程序运行的情况。

17.4.2 拥有临时结果的任务

什么是临时结果任务呢？本小节将会针对这个概念给予详细讲解。所谓的临时结果也就是工作线程在后台工作到一半时，会给出一个临时的数据信息，而这个临时的数据信息就是临时结果。

那么这个临时结果的任务有什么作用呢？举个例子来说，当后台运行一个搜索数据库的操作时，前台程序正在运行，突然想看看后台的运行情况，此时就需要执行一个临时结果的任务，将此时此刻程序运行的结果调到前台程序中来。

如何处理临时结果任务呢？方法是在工作线程中使用 `publish()` 方法将中间数据存入其中，再覆盖 `process()` 方法，存入的中间数据将被它调到事件分派线程中来。

下面列举一个实例，通过该实例，并对照上面所讲的方法，相信读者很快就可以掌握其用法。程序代码如下所示：

```
// 这段代码主要展示如何处理正在运行的程序的中间结果
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import javax.swing.SwingUtilities;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingWorker;

public class threadtest
{
    static final int WIDTH=600;
    static final int HEIGHT=800;
    JFrame frame;
    JPanel pane;
    static JTextField text1;
    private worker wor1;
    private worker wor2;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        pane.add(c,constraints);
    }
    public threadtest()
    {
        frame=new JFrame("中间结果任务测试");
        pane=new JPanel();
        frame.setContentPane(pane);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
```



```

int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
frame.setSize(WIDTH,HEIGHT);
frame.setLocation(x,y);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
frame.pack();
GridBagLayout layout=new GridBagLayout();
pane.setLayout(layout);
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.anchor=GridBagConstraints.EAST;
constraints.weightx=4;
constraints.weighty=3;
JButton button1=new JButton("运行后台计数器一");
JButton button2=new JButton("运行后台计数器二");
JButton button3=new JButton("临时结果操作一");
JButton button4=new JButton("临时结果操作二");
JTextField text1=new JTextField(10);
add(button1,constraints,1,0,1,1);
add(button3,constraints,0,1,1,1);
add(text1,constraints,2,1,1,1);
add(button2,constraints,1,2,1,1);
add(button4,constraints,0,3,1,1);
...
}
}

```

下面将给出几个按钮组件的动作事件处理，当单击第 1 个按钮组件或者第 3 个按钮组件时，会让线程在后台工作，当单击第 2 个按钮组件或者第 4 个按钮组件时，会让线程停止执行，并且将中间结果显示处理，代码如下：

```

button1.addActionListener(new ActionListener()                // 将计数工作放入工作线程，然后执行它
{
    public void actionPerformed(ActionEvent e)
    {
        (wor1 = new worker()).execute();
    }
});
button2.addActionListener(new ActionListener()                // 将计数工作放入工作线程，然后执行它
{
    public void actionPerformed(ActionEvent Event)
    {
        (wor2 = new worker()).execute();
    }
});
button3.addActionListener(new ActionListener()
// 当单击此按钮时，后台计数会停止，并且会将目前的数据显示在文本组件中
{
    public void actionPerformed(ActionEvent Event)
    {
        wor1.cancel(true);
        wor1 = null;
    }
}

```



```
});
button4.addActionListener(new ActionListener()
// 当单击此按钮时，后台计数会停止，并且会将目前的数据显示在文本组件中
{
    public void actionPerformed(ActionEvent Event)
    {
        wor2.cancel(true);
        wor2 = null;
    }
});
```

下面将创建几个内部类，这些类主要是进行内部计算功能，然后在其中取出中间结果并显示出来，代码如下：

```
private static class counter
{ // 创建一个内部类，使用其构造器赋值
    private final int sum;
    counter(int sum)
    {
        this.sum=sum;
    }
}

private class worker extends SwingWorker<Void, counter>
{ // 创建一个 worker 类，这个类继承自 SwingWorker 类
    // 所以要实现 doInBackground()和 process()方法。这个方法主要是实现一个计数功能
    protected Void doInBackground()
    {
        int sum = 0;
        while (!isCancelled())
        {
            sum++;
        }
        publish(new counter(sum)); // 使用 publish()方法将中间数据储存起来
        return null;
    }
    protected void process(List<counter> count)
    { // 这个方法主要是将中间数据从 publish()方法中提取出来，在放入到事件分派线程中去显示
        counter cou = count.get(count.size() - 1);
        text1.setText(String.format("%d", cou.sum));
    }
}

public static void main(String[] args)
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new threadtest();
        }
    });
}
```

上面程序的运行结果如图 17.2 所示。

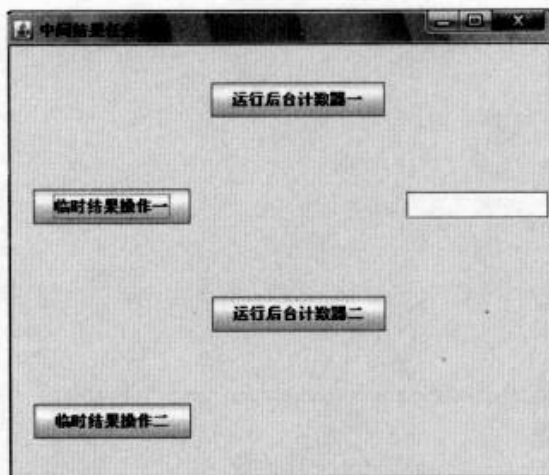


图 17.2 中间结果任务 (a)

当单击“运行后台计数器一”按钮时，sum 变量会在后台不断加 1，当单击“临时结果操作一”按钮时，文本框中会出现一个临时结果数据。同样，当单击“运行后台计数器二”按钮后，sum 变量会在后台不断加 1，当单击“临时结果操作二”按钮时，文本框中也会出现一个临时结果数据，如图 17.3 所示。

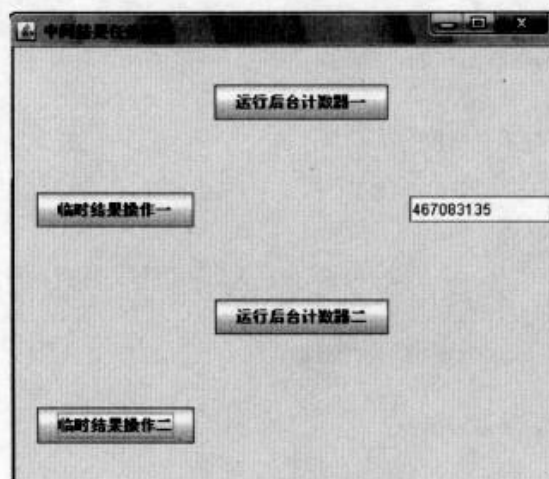


图 17.3 中间结果任务 (b)

其实中间结果主要是依靠两个方法的配合实现的，即 `publish()` 和 `process()` 方法。`publish()` 用于获取中间结果，`process()` 则是将这个结果发送到事件分派线程中去，让事件分派线程将这个结果放到前台去显示。

17.4.3 取消背景任务

如何取消一个背景任务呢？很简单，可以使用 `SwingWorker.cancel(Boolean b)` 来取消一个背景任务。如果任务已经完成了或者已经被取消了，甚至由于某种原因无法取消的话，则调用它会失败并且会弹出异常。

当调用它时，如果任务还没有启动，则任务就永远无法运行了，所以说在使用取消背景任务的方法时，一定要注意其放置的位置，从而避免了程序的混乱。下面将针对上面的实例进行修改，其具体的代码如下所示：

```
// 这段程序代码主要为读者展示如何取消背景任务
import java.awt.*;
import java.util.List;
```



```
import javax.swing.*;
public class threadtest1
{
    static final int WIDTH=600;
    static final int HEIGHT=800;
    JFrame frame;
    JPanel pane;
    static JTextField text1;
    static worker wor1;
    static worker wor2;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        ...
        // 与上述内容中相同位置的代码相似，这里不再列出
    }
    public threadtest1()
    {
        ...
        // 与上述内容中相同位置的代码相似，这里不再列出
        button1.addActionListener
        (new ActionListener()                // 将计数工作放入工作线程，然后执行它
        {
            public void actionPerformed(ActionEvent e)
            {
                (wor1 = new worker()).execute();
            }
        });
        button2.addActionListener
        (new ActionListener()                // 将计数工作放入工作线程，然后执行它
        {
            public void actionPerformed(ActionEvent Event)
            {
                (wor2 = new worker()).execute();
            }
        });
        button3.addActionListener
        (new ActionListener()                // 当单击此按钮时，后台计数会停止，并且会将目前的数据显示在文本组件中
        {
            public void actionPerformed(ActionEvent Event)
            {
                wor1.cancel(true);
                wor1 = null;
            }
        });
        button4.addActionListener
        (new ActionListener()                // 当单击此按钮时，后台计数会停止，并且会将目前的数据显示在文本组件中
        {
            public void actionPerformed(ActionEvent Event)
            {
                wor2.cancel(true);
                wor2 = null;
            }
        });
        button5.addActionListener
        (new ActionListener()
        {
```



```

        public void actionPerformed(ActionEvent Event)
        {
            wor1.cancel(true);
            wor2.cancel(true);
        }
    });
}

private static class counter
{
    private final int sum;
    counter(int sum)
    {
        this.sum=sum;
    }
}

private class worker extends SwingWorker<Void, counter>
{
    protected Void doInBackground()
    {
        int sum = 0;
        while (!isCancelled())
        {
            sum++;
        }
        publish(new counter(sum));
        return null;
    }

    protected void process(List<counter> count)
    {
        counter cou = count.get(count.size() - 1);
        text1.setText(String.format("%d", cou.sum));
    }
}

public static void main(String[] args)
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            wor1.cancel(true);
            wor2.cancel(true);
            new threadtest1();
        }
    });
}
}

```

上面的程序段的运行结果会出现异常，这是由于在执行 `threadtest()` 对象之前，系统就已经将背景任务取消了，所以会报错。读者一定要牢记取消背景任务的代码的放置位置。

17.4.4 绑定属性和状态方法

什么是绑定属性和状态方法呢？本小节将会为读者讲述其概念。先来看一下有关绑定属性的概念。`SwingWorker` 是支持绑定属性的，这个属性在与其他线程通信时可以用来观察其

状态情形。共有两个绑定属性：progress 和 state。它们可以使用在事件分派线程中的事件处理任务。开发人员可以通过实现一个 `propertyChangeListener()` 方法捕捉到这些绑定属性的变化。

progress 绑定属性的值应该在 0~100 之间。可以通过 `setProgress()` 和 `getProgress()` 方法来操作绑定属性。通过对这个属性的操作，可以得知 `SwingWorker` 对象目前的状态值。

state 绑定属性的值变化反映了 `SwingWorker` 对象在它的生命周期中的变化过程。它的属性值有：DONE、PENDING、STARTED。

- DONE: `doInBackground` 方法完成之后执行 `SwingWorker`。
- PENDING: 初始的 `SwingWorker` 状态。
- STARTED: 在调用 `doInBackground` 之前启动 `SwingWorker`。

状态方法用于报告背景任务目前的状态情形。有兴趣的读者可以查阅相关的资料，自行举例练习。

17.5 本章小结

本章主要是为读者讲述了有关 Swing 图形开发程序中的多线程知识，使读者清楚如何能在 Swing 开发的程序中，尽量让整个程序的运行速度发挥到极致。当然，本章限于篇幅问题，还有很多细小的知识点没有涉及到，希望读者能够以本章为基础多多参阅其他的相关资料。

17.6 本章习题

1. 设计一个程序，在程序中有一个文本组件，用来显示从 1~1000000 的值的和，另外，另一个文本组件用来显示从 1~100 的和，并对比将第 1 个文本组件运行放在后台和不放在后台的结果。

要求：其最终结果如图 17.4 所示。



图 17.4 本章习题 1

2. 设计一个程序，在程序中有一个文本框组件、两个按钮组件，其中一个按钮组件用来启动从 1 到无穷大的和的计算程序，另一个按钮组件用来获得临时结果。

要求：上面的程序运行后，单击“开始计算”按钮，再单击“获得临时的结果”按钮，文本框中会出现临时的结果，如图 17.5 所示。

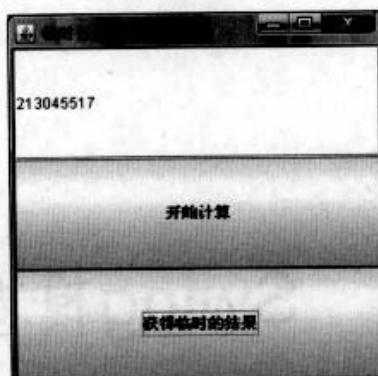


图 17.5 本章习题 2

3. 设计一个程序，程序中有两个按钮，每个按钮都用于打开一个顶层容器，第 1 个顶层容器延迟打开 10000ms。当先单击第 1 个按钮组件后再单击第 2 个按钮组件，尝试让第 2 个窗口的出现不会因为第 1 个窗口没有出现而停滞。

4. 请利用事件分派线程的原理实现上题的功能。

第18章 Swing模型架构

本章主要介绍有关 Swing 程序开发的模型架构方面的知识。首先会介绍传统模型架构的体系结构，并且会提出传统模型架构的弊端，随后，将针对这些弊端，提出目前所使用的新型的模型架构。在讲解新型的模型架构的同时，也会将新型的模型架构的特点和缺点一一列举出来，使读者能够充分了解它，尽量避免这种模型架构的缺点在自己开发的程序中出现。另外，由于本章是一章纯理论的章节，所以希望读者能够仔细的阅读，尽量弄懂每一个知识点的细节知识。

18.1 传统的 MVC 设计模式

目前，几乎所有的程序开发人员都很清楚 Swing 图形开发所采用的设计模式是模型与视图分离的 MVC 设计模式，那什么是 MVC 设计模式呢？本节将给出详细讲解，同时还会为读者讲述其优缺点，从而让读者真正地理解传统的 MVC 设计模式的设计理念，以及为何这种传统的 MVC 设计模式会被淘汰。

由于本章的理论性比较强，为了能让读者充分地理解每一个概念，下面先为读者讲述模式到底是什么。可能此时不一定会有很多人能够知道模式的真正含义。所谓的模式，其实就像一个战场上的司令员，它能够聚集战士们的斗志，统一对敌。而开发人员也是在一个模式的规则下共同完成一个开发项目。下面将通过比较一下模式的示意图和司令员的示意图，让读者真正理解模式的含义。司令员示意图如图 18.1 所示。

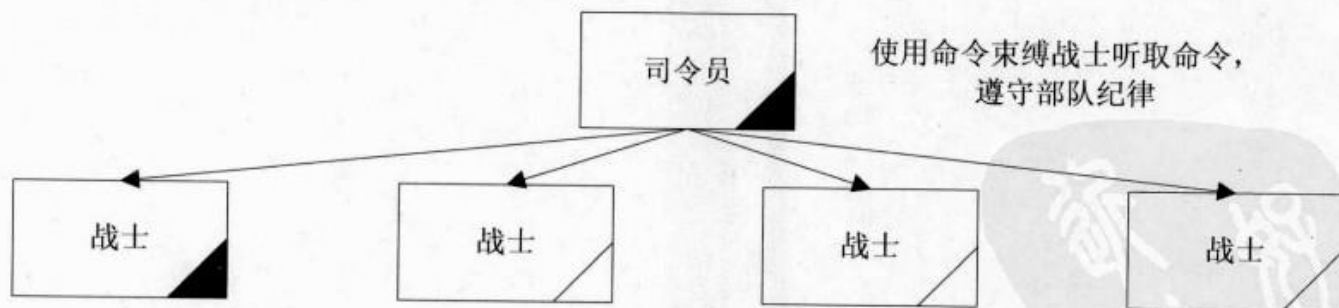


图 18.1 司令员示意图

下面再为读者展示一下模式的示意图，如图 18.2 所示。

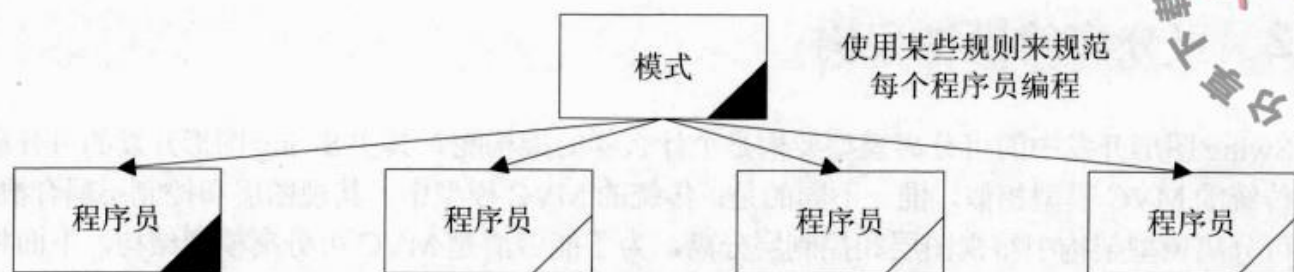


图 18.2 模式示意图

从上面的示意图和理论说明可以知道，其实模式就是一类事物处理的规范。所谓的设计模式也就是设计规范，通俗地说就是设计所必须遵循的规范。

接下来为读者讲述什么是 MVC，其实 MVC 是一组英文的缩写，其全名是 Model-View-Controller，中文名称为“模型-视图-控制器”，所以，MVC 就分为三个部分：模型（Model）、视图（View）、控制器（Controller）。这三个部分中任意一个部分发生变化都会引起另外两个发生变化，使用专业的语言描述就是这三个部分具有紧耦合性。下面将给出三者之间的关系示意图，如图 18.3 所示。

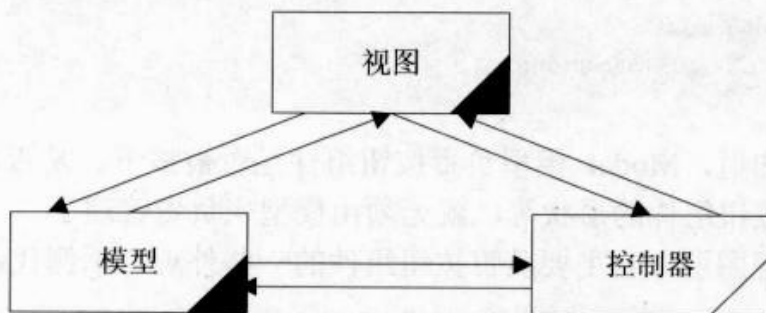


图 18.3 MVC 三部分关系示意图

由于 MVC 分成了三个部分，所以在 MVC 模型中将按照此三部分分成三层，分别是视图层、模型层、控制层。下面将针对这三层给予详细介绍，通过介绍让读者能够清楚地了解此三层是如何形成一个 MVC 模型的。

- 模型层：这里的模型通常指业务逻辑的处理和数据的存储。模型层主要的作用是接受控制层发送过来的客户请求，处理后的数据再通过控制层返回到视图层，也就是返回给用户界面。
- 视图层：这一层用来向用户展现自己所需要的数据，它是用户和系统进行交互的界面。举个简单的例子，在学校信息系统软件中，经常会有一项功能就是查询，当用户单击查询某些资料时，系统会查询数据库，再将结果返回给用户的软件界面，而这个返回给用户所需数据的软件界面就是视图层。
- 控制层：这一层其实就像一个事务处理中间站，它是接受视图层发送过来的用户请求数据，然后再根据用户请求的内容选择发送给哪个模型层，最后将请求发送给所选的模型层，等模型层处理好数据后，再将数据返回给视图层。

通过上面对三个层面的介绍，可以看出三层中任意一层的变化都会引起其他两层不同的变化。通常，开发人员并不会直接和模型层、视图层以及控制层接触，因为这些层都被隐藏在组件中，这些组件的 MVC 中的视图层和控制层有一定的紧密关联，而在传统的 MVC 模式中却硬生生地将之分开。有经验的开发人员一定会认为这样做是很不严格的，所以为了能够解决这个问题，随后就出现了一个新型的模式架构——可分离模型架构。

18.2 可分离的模型架构

Swing 图形开发中的可分离模型架构是个什么样的架构呢？其实 Swing 图形开发的可分离模型同传统的 MVC 模型相似，惟一不同的是：传统的 MVC 模型中，其视图层和控制层耦合很紧，而在可分离模型架构中将视图层和控制层分离，为了能够清楚 MVC 可分离模型结构，下面将给出实例，并且给予分析。实例中以按钮组件为分析对象，其主要是针对按钮组件的创建。

ButtonModel 类属于模型层，所以它基本上负责 Button 按钮的自身状态信息等。代码如下所示：

```
// 按钮被按下的状态或者被选择的状态
private boolean boolPressed = false;
public boolean isPressed()
{
    return boolPressed;
}
public void setPressed(boolean boolPressed)
{
    this.boolPressed = boolPressed;
    fireChangeEvent(new ChangeEvent(button));
}
```

由以上代码可以知道，Model 模型负责按钮组件是否被按下、是否被选择等，而对于按钮组件的事件处理、按钮组件的形状等，就无须由模型来负责管理了。

ButtonUI 类属于视图层，它主要负责按钮组件的一些外观，示例代码如下：

```
// 主要是描述按钮组件的外观、颜色等
public void update(Button button, Graphics graphics)
{
}
public void paint(Button button, Graphics graphics)
{
    Dimension dimension = button.getSize();
    Color color = button.getBackground();
    graphics.setColor(color);
    graphics.fillRect(0, 0, dimension.width, dimension.height);
}
```

由以上代码可以看出视图层主要是负责外观。而真正的处理事件则交给控制层的监听器去处理。不过视图层会将事件处理给予定义，代码如下所示：

```
// 主要是定义监听事件，通过监听事件来控制按钮的行为
private static ButtonUIListener buttonuilibtenser = null;
public void installUI(Button button)
{
    button.addMouseListener(buttonuilibtenser);
    button.addMouseMotionListener(buttonuilibtenser);
    button.addChangeListener(buttonuilibtenser);
}
public void uninstallUI(Button button)
{
    button.removeMouseListener(buttonuilibtenser);
    button.removeMouseMotionListener(buttonuilibtenser);
    button.removeChangeListener(buttonuilibtenser);
}
```


通过以上实例,相信大家应该能够更加理解 MVC 设计模式的真正含义,其实,真正能够将这种设计理念使用到实际是很困难的,不过希望读者能够在理解这个概念的同时,也能尝试将 MVC 设计模式运用到自己的程序中去。

18.3 本章小结

本章通过理论和实例的方法为读者详细讲述了有关 Swing 图形开发中的设计模式,其中包括传统的 MVC 模式和新型的可分离模型两种。无论是传统的 MVC 模式,还是新型的可分离模式,其实被称为 MVC 模式。MVC 模式有自己的特点和缺点。MVC 的最大的优点就是其分层的思想以及代码的可重用性提高了。最大缺点就是对开发人员的要求很高,并且测试会比较困难。有兴趣的读者可以以本章为基础,查阅其他相关的资料进行深入学习。

18.4 本章习题

1. 什么是设计模式?
2. 什么是 MVC?
3. 什么是可分离模型架构?
4. 传统的 MVC 和可分离的 MVC 有什么不同?

第19章 Swing的其他特性

本章主要介绍一些实际开发中遇到的常用知识，这些知识对于开发人员来说是非常重要的，由于这些知识不属于某个具体的系统模块，所以只能将它们单独列举出来进行讲述。这些知识点包括边框、动作、辅助技术、绑定技术、拖曳技术、如何使用图标、如何使用焦点子系统等，它们都是实际开发中使用频繁的知识点。为了能够在短时间内迅速掌握这些技巧，本章将会给予大量的程序实例，以帮助读者了解这些知识在实际开发中是如何被使用的。

19.1 如何在 Swing 组件中使用 HTML

在 Swing 图形开发中，许多组件都要在其上显示一些文本来表示组件的用处或者是呈现某些内容给用户。但是这些组件上的文本多半是单一字体和单一颜色，如果将 HTML 文档添加到 Swing 组件上的话，那么其文本将会变的很有特色。其实，HTML 就相当于一个格式化的工具，它可以将一些文本的字体、颜色、显示等进行一定的包装，然后呈现在用户眼前。

HTML 的格式化功能可以在几乎所有的 Swing 按钮、菜单项、标签、选项卡等中使用，如果读者有过编写或者研究 HTML 的经历，应该知道 HTML 文档是以“<html>”和“</html>”为开头标志和结尾标志的，同理，在 Swing 组件中也是以“<html>”和“</html>”为文本开头标志和结尾标志的。下面先来看一个按钮组件上的文本是如何使用 HTML 文本来表示的，代码如下：

```
 JButton button=new JButton("<html><b>HTML 文本组件</b></html>");
```

上面的代码表示一个按钮组件，其文本是以 HTML 文档来表示的。其运行的结果如图 19.1 所示。

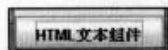


图 19.1 一个带 HTML 文本的按钮组件

如何在组件上使用 HTML 文档作为其文本呢？其实很简单，使用 HTML 语句来代替文本字符串即可。下面将给出一个实例，通过该实例，读者能够清楚地了解如何在 Swing 组件中使用 HTML 文本来代替普通文本。该实例用于创建 5 个按钮组件，这个 5 个按钮组件上的文本将会通过不同的 HTML 文档，使其展现出不同的外观，其具体的程序代码如下所示：

```
// 这个实例主要是创建 5 个按钮组件，这 5 个按钮组件上的文本将会通过不同 HTML 文档展现出不同的外观
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JButton;
```



```

import javax.swing.JFrame;
import javax.swing.JPanel;
public class test1
{
    static final int WIDTH=700;
    static final int HEIGHT=400;
    public static void main(String[] args)
    {
        JFrame frame=new JFrame("<html><b>HTML 文本测试程序</b></html>");
        // 创建一个顶层容器类对象，并且其标题使用粗体
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setSize(WIDTH, HEIGHT);
        frame.setVisible(true);
        frame.pack();
        JButton button1=new JButton("<html><b>第一个按钮组件</b><br>"+<font color=red>测试一</font></html>");
        // 创建 5 个按钮组件，每个按钮组件上面的文字外观、颜色不一样
        JButton button2=new JButton("<html><b>第二个按钮组件</b><br>"+<font color=blue>测试二</font></html>");
        JButton button3=new JButton("<html><b>第三个按钮组件</b><br>"+<font color=green>测试三</font></html>");
        JButton button4=new JButton("<html><b>第四个按钮组件</b><br>"+<font color=orange>测试四</font></html>");
        JButton button5=new JButton("<html><b>第五个按钮组件</b><br>"+<font color=grey>测试五</font></html>");
        JPanel panel=new JPanel();
        panel.add(button1);
        panel.add(button2);
        panel.add(button3);
        panel.add(button4);
        panel.add(button5);
        frame.setContentPane(panel);
    }
}

```

上面程序代码的运行结果如图 19.2 所示。

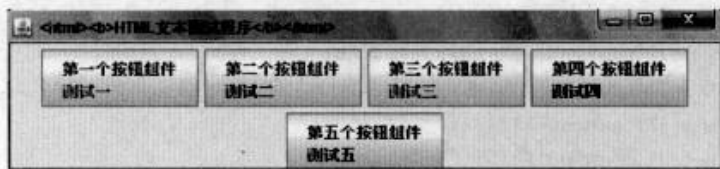


图 19.2 按钮组件和窗口名称使用 HTML

下面再给出一个有关如何在菜单项中使用 HTML 的实例，其程序代码如下所示：

```

// 这段程序代码主要为读者展示如何为一个图形界面添加各种不同的格式
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test2
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;

```



```

JPopupMenu pop;
JMenuItem item2;
JFrame f;
JMenuItem item1;
JPanel p;
JTabbedPane tp;
public test2()
{
    // 设置顶层窗口标题的颜色
    f=new JFrame("<html><font color=red>星火纺织厂信息系统</font></html>");
    JMenuBar menubar1=new JMenuBar();
    tp=new JTabbedPane();
    p=new JPanel();
    f.setJMenuBar(menubar1);
    BorderLayout bord = new BorderLayout();
    p.setLayout(bord);
    f.add(p);
    p.add("Center",tp);
    JMenu menu1=new JMenu("<html><font color=red>职工信息系统</font></html>");
    JMenu menu2=new JMenu("<html><font color=red>中层干部信息系统</font></html>");
    JMenu menu3=new JMenu("<html><font color=red>工资系统</font></html>工资系统");
    JMenu menu4=new JMenu("<html><font color=red>工资系统查询系统</font></html>");
    JMenu menu5=new JMenu("登录系统");
    JMenu menu6=new JMenu("帮助系统");
    menu1.setMnemonic('Z');
    menu2.setMnemonic('C');
    menu3.setMnemonic('G');
    menu4.setMnemonic('F');
    menu5.setMnemonic('D');
    menu6.setMnemonic('H');
    menubar1.add(menu1);
    menubar1.add(menu2);
    menubar1.add(menu3);
    menubar1.add(menu4);
    menubar1.add(menu5);
    menubar1.add(menu6);
    Item1=new JMenuItem("磨砂分厂职工信息");
    item2=new JMenuItem("纺纱分厂职工信息");
    JMenuItem item3=new JMenuItem("人事部职工信息");
    JMenuItem item4=new JMenuItem("财务部职工信息");
    JMenuItem item5=new JMenuItem("材料科职工信息");
    JMenuItem item6=new JMenuItem("销售科职工信息");
    JMenuItem item7=new JMenuItem("成品车间职工信息");
    JMenuItem item8=new JMenuItem("分厂厂长信息");
    JMenuItem item9=new JMenuItem("部门经理信息");
    JMenuItem item10=new JMenuItem("职工工资信息");
    JMenuItem item11=new JMenuItem("领导工资信息");
    JMenuItem item12=new JMenuItem("按照姓名查询");
    JMenuItem item13=new JMenuItem("按照工号查询");
    JMenuItem item14=new JMenuItem("添加登录用户名");
    JMenuItem item15=new JMenuItem("版本信息");
    JMenuItem item16=new JMenuItem("帮助信息");
    item1.setAccelerator( KeyStroke.getKeyStroke('M', java.awt.Event.CTRL_MASK, false) );
    item2.setAccelerator( KeyStroke.getKeyStroke('F', java.awt.Event.CTRL_MASK, false) );
    item3.setAccelerator( KeyStroke.getKeyStroke('R', java.awt.Event.CTRL_MASK, false) );
    item4.setAccelerator( KeyStroke.getKeyStroke('C', java.awt.Event.CTRL_MASK, false) );

```



```

menu1.add(item1);
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu1.add(item4);
menu1.add(item5);
menu1.add(item6);
menu1.add(item7);
menu2.add(item8);
menu2.add(item9);
menu3.add(item10);
menu3.add(item11);
menu4.add(item12);
menu4.add(item13);
menu5.add(item14);
menu6.add(item15);
menu6.add(item16);
// 设置三个按钮组件, 这三个按钮组件上的文本分别设置成蓝色
JButton button1 = new JButton("<html><font color=blue>磨砂分厂职工信息</font></html>");
JButton button2 = new JButton("<html><font color=blue>纺纱分厂职工信息</font></html>");
JButton button3 = new JButton("<html><font color=blue>人事部职工信息</font></html>");
JToolBar bar = new JToolBar();
bar.add(button1);
bar.add(button2);
bar.add(button3);
p.add("North",bar);
f.setVisible(true);
f.setSize(WIDTH,HEIGHT);
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);
button1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        info con1=new info();
        JPanel con2 = new JPanel ();
        JPanel con3 = new JPanel ();
        JPanel con4 = new JPanel ();
        JPanel con5 = new JPanel ();
        tp.addTab("con1", con1);
        tp.setEnabledAt(0,true);
        // 设置 JTabbedPane 组件上的每一页的文本颜色为绿色
        tp.setTitleAt(0,"<html><font color=green>一车间信息</font></html>");
        tp.addTab ("con2", con2);
        tp.setEnabledAt (1, true);
        tp.setTitleAt (1,"<html><font color=green>二车间信息</font></html>");
        tp.addTab ("con3", con3);
        tp.setEnabledAt (2, true);
        tp.setTitleAt (2,"<html><font color=green>三车间信息</font></html>");
        tp.addTab ("con4", con4);
        tp.setEnabledAt(0,true);
    }
}

```



```

        tp.setTitleAt(3,"<html><font color=green>四车间信息</font></html>");
        tp.addTab("con5", con5);
        tp.setEnabledAt(4,true);
        tp.setTitleAt(4,"<html><font color=green>五车间信息</font></html>");
        tp.setPreferredSize(new Dimension(500,200));
        tp.setTabPlacement(JTabbedPane.TOP);
        tp.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
    }
    });
    ...
}
public static void main(String args[])
{
    new test2();
}
}
class info extends JPanel
{
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    info()
    {
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
        // 创建 8 个标签, 并且将每个标签的文本颜色设置成红色
        JLabel name=new JLabel("<html><font color=red>姓名</font></html>");
        JLabel code=new JLabel("<html><font color=red>工号</font></html>");
        JLabel sexy=new JLabel("<html><font color=red>性别</font></html>");
        JLabel age=new JLabel("<html><font color=red>年龄</font></html>");
        JLabel birthday=new JLabel("<html><font color=red>出生年月</font></html>");
        JLabel address=new JLabel("<html><font color=red>家庭地址</font></html>");
        JLabel cj=new JLabel("<html><font color=red>车间</font></html>");
        JLabel zw=new JLabel("<html><font color=red>职位</font></html>");
        final JTextField codeinput=new JTextField(10);
        final JTextField nameinput=new JTextField(10);
        final JTextField sexyinput=new JTextField(10);
        final JTextField ageinput=new JTextField(10);
        final JTextField birthdayinput=new JTextField(10);
        final JTextField addressinput=new JTextField(10);
        final JTextField gradeinput=new JTextField(10);
        final JTextField majorinput=new JTextField(10);
        // 设置标签的文本颜色为蓝色
        JLabel title=new JLabel("<html><font color=blue>基本信息</font></html>");
        JButton additionbutton=new JButton("<html><font color=blue>添加</font></html>");
        GridBagConstraints constraints=new GridBagConstraints();
        constraints.fill=GridBagConstraints.NONE;
        constraints.weightx=4;
        constraints.weighty=6;
        add(title,constraints,0,0,4,1);
        add(name,constraints,0,1,1,1);
        // 使用网格组布局添加控件
    }
}

```



```
add(code,constraints,0,2,1,1);
add(sexy,constraints,0,3,1,1);
add(age,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(codeinput,constraints,1,2,1,1);
add(sexyinput,constraints,1,3,1,1);
add(ageinput,constraints,1,4,1,1);
add(birthday,constraints,2,1,1,1);
add(address,constraints,2,2,1,1);
add(cj,constraints,2,3,1,1);
add(zw,constraints,2,4,1,1);
add(birthdayinput,constraints,3,1,1,1);
add(addressinput,constraints,3,2,1,1);
add(gradeinput,constraints,3,3,1,1);
add(majorinput,constraints,3,4,1,1);
}
```

上面程序代码的运行结果如图 19.3 所示。

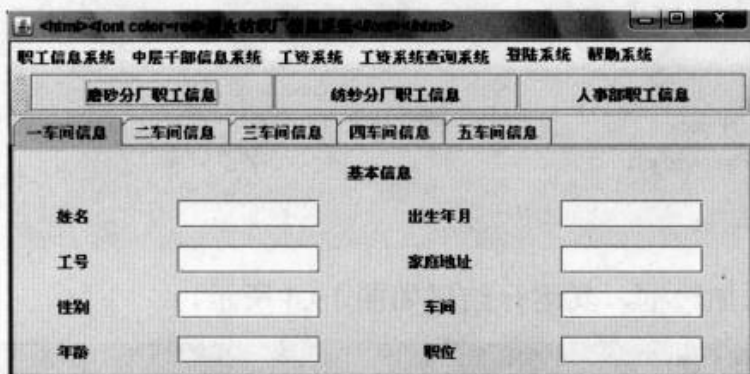


图 19.3 菜单项和选项卡使用 HTML 作为文本

从以上两个程序实例中可以看出，利用 HTML 文档替代文本是非常简单的事情，只要将普通文本换成 HTML 文本即可。

19.2 如何使用边框

本节将详细讲述边框的知识。如何让组件使用 Swing 库中边框呢？通过对比设置了边框对象和没有设置边框对象的两个实例，使读者能够清楚地了解边框的作用。而后将进一步介绍如何设计一个具有个人风格的自定义边框对象。

19.2.1 如何使用 Swing 中的边框

什么是边框？边框就是一个组件的边界。虽然边框不是控件，但是边框不仅可以对组件起到一个美观的作用，而且还可以提供标题和组件周围的空白控件。如果要想在一个组件周围设置边框，可以使用 `setBorder()` 方法，这个方法中的参数可以使用 `BorderFactory` 类所创建的边框模板。下面将通过两个实例来对比一下设置了边框和没有设置边框的区别。没有设置边框的框架的程序代码如下所示：

```
// 这段程序代码主要是为读者展示不添加边框的界面
import java.awt.Color;
```



```
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class test3
{
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public static void main(String[] args)
    {
        JFrame frame=new JFrame("边框测试程序");
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setSize(WIDTH, HEIGHT);
        frame.setVisible(true);
        frame.pack();
        JPanel pane=new JPanel();
        frame.setContentPane(pane);
    }
}
```

上面的程序没有添加边框，其运行结果如图 19.4 所示。

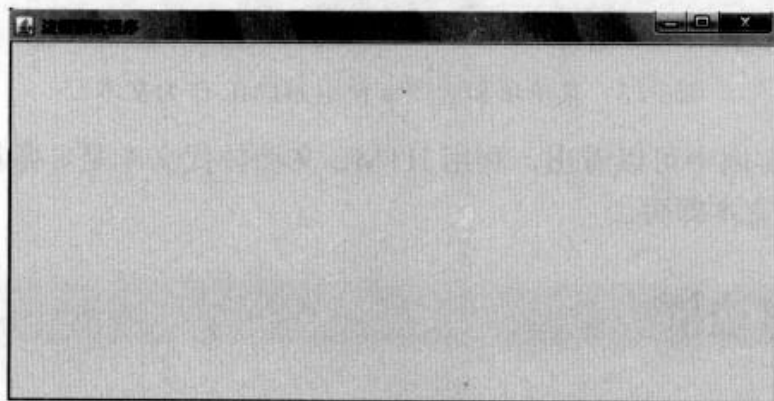


图 19.4 没有添加边框的 JPanel

下面将给出一段添加了边框的程序代码，如下所示：

```
// 这段程序代码主要是为读者展示如何为容器添加边框
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class test4
{
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public static void main(String[] args)
    {
```



```

JFrame frame=new JFrame("边框测试程序");
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
frame.setLocation(x,y);
frame.setSize(WIDTH, HEIGHT);
frame.setVisible(true);
frame.pack();
JPanel pane=new JPanel();
frame.setContentPane(pane);
pane.setBorder(BorderFactory.createLineBorder(Color.red));
// 设置内容面板的边框颜色为红色
}
}

```

上面程序代码的运行结果如图 19.5 所示。

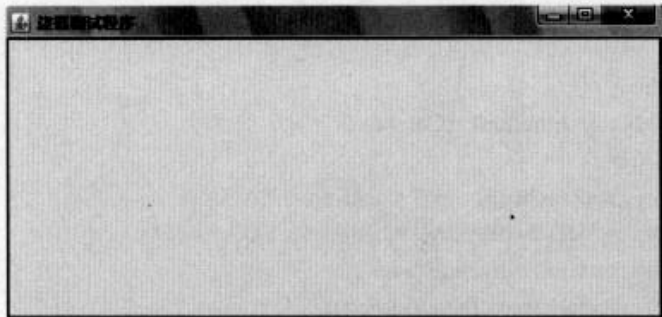


图 19.5 添加了边框的 JPanel

通过上面的实例可以知道，为面板添加边框是非常简单的。接下来看一下 Swing 中提供给开发人员的边框是什么样的。在这里将通过程序代码的方式进行说明：

// 这段程序代码主要是为读者展示如何为各种各样的组件添加不同的边框

```

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;
public class test5 extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;

```



```
public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
{
    constraints.gridx=x;
    constraints.gridy=y;
    constraints.gridwidth=w;
    constraints.gridheight=h;
    add(c,constraints);
}

public test5()
{
    JFrame frame=new JFrame("边框测试程序");
    Toolkit kit=Toolkit.getDefaultToolkit();
    Dimension screenSize=kit.getScreenSize();
    int width=screenSize.width;
    int height=screenSize.height;
    int x=(width-WIDTH)/2;
    int y=(height-HEIGHT)/2;
    frame.setLocation(x,y);
    frame.setSize(WIDTH, HEIGHT);
    frame.setVisible(true);
    frame.pack();
    frame.setContentPane(this);
    Border b1=BorderFactory.createLineBorder(Color.red);
    // 设置 10 种不同的边框模式
    Border b2=BorderFactory.createEtchedBorder(EtchedBorder.RAISED);
    Border b3=BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
    Border b4=BorderFactory.createRaisedBevelBorder();
    Border b5=BorderFactory.createLoweredBevelBorder();
    Border b6=BorderFactory.createTitledBorder("边框设计");
    Border b7=BorderFactory.createTitledBorder(b1,"边框设计");
    Border b8=BorderFactory.createTitledBorder(b3,"边框设计");
    Border b9=BorderFactory.createTitledBorder(b5,"边框设计");
    Border b10=BorderFactory.createCompoundBorder(b4,b5);
    JLabel lable1=new JLabel("按钮一");
    JLabel lable2=new JLabel("按钮二");
    JLabel lable3=new JLabel("按钮三");
    JLabel lable4=new JLabel("按钮四");
    JLabel lable5=new JLabel("按钮五");
    JLabel lable6=new JLabel("按钮六");
    JLabel lable7=new JLabel("按钮七");
    JLabel lable8=new JLabel("按钮八");
    JLabel lable9=new JLabel("标题");
    JTextField text1=new JTextField(10);
    JTextField text2=new JTextField(10);
    JTextField text3=new JTextField(10);
    JTextField text4=new JTextField(10);
    JTextField text5=new JTextField(10);
    JTextField text6=new JTextField(10);
    JTextField text7=new JTextField(10);
    JTextField text8=new JTextField(10);
    setBorder(b1);
    lable1.setBorder(b2);
    lable2.setBorder(b3);
    lable3.setBorder(b4);
    lable4.setBorder(b5);
    lable5.setBorder(b6);
```



```

lable6.setBorder(b7);
lable7.setBorder(b8);
lable8.setBorder(b9);
lable9.setBorder(b10);
text1.setBorder(b2);
text2.setBorder(b3);
text3.setBorder(b4);
text4.setBorder(b5);
text5.setBorder(b6);
text6.setBorder(b7);
text7.setBorder(b8);
text8.setBorder(b9);
GridBagLayout lay=new GridBagLayout();
setLayout(lay);
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=4;
constraints.weighty=6;
add(lable9,constraints,0,0,4,1);
add(lable1,constraints,0,1,1,1);
add(lable2,constraints,0,2,1,1);
add(lable3,constraints,0,3,1,1);
add(lable4,constraints,0,4,1,1);
add(lable5,constraints,1,1,1,1);
add(text1,constraints,1,2,1,1);
add(text2,constraints,1,3,1,1);
add(text3,constraints,1,4,1,1);
add(text4,constraints,2,1,1,1);
add(text5,constraints,2,2,1,1);
add(lable6,constraints,2,3,1,1);
add(lable7,constraints,2,4,1,1);
add(text6,constraints,3,1,1,1);
add(text7,constraints,3,2,1,1);
add(text8,constraints,3,3,1,1);
add(lable8,constraints,3,4,1,1);
}
public static void main(String[] args)
{
    new test5();
}
}

```

上面程序代码的运行结果如图 19.6 所示。

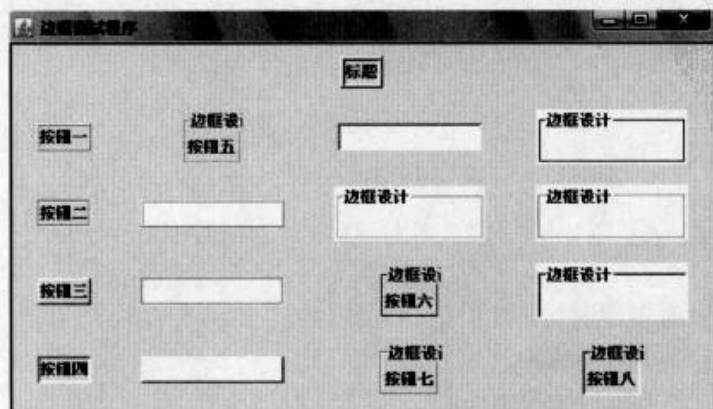


图 19.6 各种类型的边框展示

上面的实例展示了几种不同的边框，这些边框类型是否很熟悉呢？是否经常在某些程序中见到呢？其实使用 Swing 定义的边框主要是对 `BorderFactory` 类的方法的熟练运用，所以希望读者能够多多练习。

19.2.2 如何创建自定义边框

上一节主要讲述了如何使用 Swing 中定义的方法设置边框，本节将进一步讲述如何设置具有个人风格的自定义边框。

如果在 Swing 提供的众多边框模型中没有自己所需要的边框，那么可以自定义边框。一般来说，常用的方法是创建抽象接口 `Abstract Border` 的子类，在创建其子类时，必须实现子类的构造器和两个没有被实现的方法。

- `paintBorder`: 包含了组件用来绘制边框的代码。
- `getBorderInsets`: 指定了边框绘制自身所需要的空间。

下面将给予详细分析，抽象接口如下所示：

```
public interface Border
{
    // 这是画出组件边框的地方
    void paintBorder(Component c, Graphics g, int x, int y, int width, int height);
    // 这是定义边框边界的地方，组件可以根据该信息安排它的内容
    Insets getBorderInsets(Component c);
    // 边框的背景是不是透明的？不是透明的要负责画出边框的背景，是透明的话可使用组件的背景
    boolean isBorderOpaque();
}
```

这里将给出一个实例，通过实例使读者对自定义边框有一个总体上的认识，实例代码如下：

```
// 这段程序代码为读者展示如何自定义一个边框
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Insets;
import java.awt.Toolkit;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.Border;

public class test6
{
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public static void main(String[] args)
    {
        JFrame frame=new JFrame("边框测试程序");
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
```



```
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
frame.setLocation(x,y);
frame.setSize(WIDTH, HEIGHT);
frame.setVisible(true);
frame.pack();
JPanel pane=new JPanel();
frame.setContentPane(pane);
RedLineBorder border=new RedLineBorder();
pane.setBorder(border);
}
}
class RedLineBorder implements Border
{
    public void paintBorder(Component c, Graphics g, int x, int y, int width, int height)
    {
        g.setColor(Color.red);           // 设置为红色
        g.drawRect(x,y, width, height);   // 画出边框
    }
    public Insets getBorderInsets(Component c)
    {
        return new Insets(1,1,1,1);       // 四周都是 1
    }
    public boolean isBorderOpaque()
    {
        return false;                     // 背景透明
    }
}
```

上面程序的运行结果如图 19.7 所示。

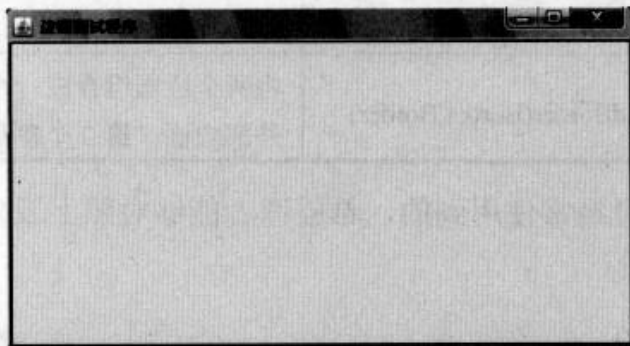


图 19.7 自定义边框

上例很简单，但是此例的主要目的就是展示如何自定义边框，希望读者能够自行举例加以练习。

19.2.3 边框组件的常用 API

在前面的实例中出现了大量的有关边框类的 API，API 的说明如表 19.1 所示。

表 19.1 边框类的 API

方法	说明
Border createLineBorder(Color)	创建一个线条边框，第 1 个参数用于指定线条颜色，第 2 个参数用于指定线宽
Border createLineBorder(Color,int)	
Border createEtchedBorder()	创建一个铭刻边框
Border createEtchedBorder(Color,Color)	
Border createEtchedBorder(int)	
Border createEtchedBorder(int,Color,Color)	
Border createLoweredBevelBorder()	创建一个能够表示组件凹入到周围空间的边框
Border createRaisedBevelBorder()	创建一个能够表示组件突出到周围空间的边框
Border createBevelBorder(int ,Color,Color)	创建一个突出或者凹入的斜面
Border createBevelBorder(int ,Color,Color,Color,Color)	
Border creatEmptyBorder()	创建一个不可见的边框
Border createEmptyBorder(int,int,int,int)	边框将不会占据空间
MatteBorder createMatteBorder(int,int,int,int,Color)	创建一个带衬边的边框，int 指边框在使用它的组件四周占据的空间
MatteBorder createMatteBorder(int,int,int,int,icon)	
itledBorder createTitleBorder(String)	创建一个带标题的边框，字符串参数指定要显示的标题。边框参数指定要与标题同时显示的边框；TitleBorder 参数指定了标题与边框的相对位置，可选参数 int 依次指定标题的位置和对齐方式
TitleBorder createTitleBorder(Border)	
TitleBorder createTitleBorder(Border,String)	
TitleBorder createTitleBorder(Border,String,int,int)	
TitleBorder createTitleBorder(Border,String,int,int,Font)	
TitleBorder createTitleBorder(Border,String,int,int,Font)	
CompoundBorder createCompoundBorder(Border,Border)	将两个边框组合成一个边框。第 1 个参数指定了外部边框，第 2 个参数指定了内部边框

上面的 API 是开发人员经常使用到的，希望读者能够对照上表多多练习，这样才能真正掌握设计边框的技巧和方法。

19.3 如何使用图标

图标用于代表一定的含义，所以在很多组件中会采用图标来代替文本来表示组件的意义。实际上，图标就是一个实现了 Icon 接口的对象。在实际开发中，开发人员常常使用 ImageIcon 类来实现 Icon 接口，而 ImageIcon 类可以根据 GIF、JPEG 或者 PNG 图像绘制图标。为了能够在组件上使用图标，必须首先创建 ImageIcon 对象，下面以表格的形式给出 ImageIcon 类的构造器，如表 19.2 所示。

表 19.2 ImageIcon 类的构造器

构造器	说明
ImageIcon()	创建一个未初始化的图像图标
ImageIcon(byte[] imageData)	根据字节数组创建一个 ImageIcon, 这些字节读取自一个包含受支持图像格式的图像文件
ImageIcon(byte[] imageData, String description)	根据字节数组创建一个 ImageIcon, 这些字节读取自一个包含受支持图像格式的图像文件
ImageIcon(Image image)	根据图像对象创建一个 ImageIcon
ImageIcon(Image image, String description)	根据图像创建一个 ImageIcon
ImageIcon(String filename)	根据指定的文件创建一个 ImageIcon
ImageIcon(String filename, String description)	根据指定的文件创建一个 ImageIcon
ImageIcon(URL location)	根据指定的 URL 创建一个 ImageIcon
ImageIcon(URL location, String description)	根据指定的 URL 创建一个 ImageIcon

下面将给出一个实例, 使读者能够熟悉如何在组件中使用图标, 其程序代码如下所示:

```
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Insets;
import java.awt.Toolkit;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.border.Border;
public class test7
{
    static final int WIDTH=600;
    static final int HEIGHT=300;
    public static void main(String[] args)
    {
        JFrame frame=new JFrame("边框测试程序");
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setSize(WIDTH, HEIGHT);
        frame.setVisible(true);
        frame.pack();
        JPanel pane=new JPanel();
        frame.setContentPane(pane);
        RedLineBorder border=new RedLineBorder();
        pane.setBorder(border);
        // 创建了 5 个图标组件对象
        ImageIcon icon1=new ImageIcon("d:/11.gif","picture");
        ImageIcon icon2=new ImageIcon("d:/111.gif","picture");
```



```
ImageIcon icon3=new ImageIcon("d:/cash_sleep.gif","picture");
ImageIcon icon4=new ImageIcon("d:/cash_walk.gif","picture");
ImageIcon icon5=new ImageIcon("d:/btn_submit_registration.gif","picture");
// 将 5 个图标对象分别加入到 5 个标签中
JLabel label1=new JLabel(icon1);
JLabel label2=new JLabel(icon2);
JLabel label3=new JLabel(icon3);
JLabel label4=new JLabel(icon4);
JLabel label5=new JLabel(icon5);
pane.add(label1);
pane.add(label2);
pane.add(label3);
pane.add(label4);
pane.add(label5);
}
}
class RedLineBorder implements Border
{
    public void paintBorder(Component c, Graphics g, int x, int y, int width, int height)
    {
        g.setColor(Color.red);           // 设置为红色
        g.drawRect(x,y, width, height);   // 画出边框
    }
    public Insets getBorderInsets(Component c)
    {
        return new Insets(1,1,1,1);
        // 4 周都是 1
    }
    public boolean isBorderOpaque()
    {
        return false;                     // 背景透明
    }
}
```

上面程序代码的运行结果如图 19.8 所示。



图 19.8 如何使用图标

上面的程序中有 5 个标签组件，每个标签组件都使用了图标。其实，很多组件都可以使用标签，例如在日常开发中最常用的是按钮组件，它可代替文字来使用。

19.4 如何使用动作

在很多软件中，当单击一个菜单项、一个按钮、输入用户密码后回车等，这些都涉及到了一个概念——动作。其实，动作事件处理在前面的章节中也曾讲述过，是通过针对组件进行添加监听器来处理事件的，然而本节将会通过另一种方法——Action 类来为读者讲述如何

处理动作事件。

要创建一个 Action 对象，通常要创建一个抽象类 AbstractAction 的子类并且要初始化它。在创建的子类中，必须要实现 actionPerformed 方法来响应动作事件的发生。为了让读者能够充分地理解如何使用 Action 类创建动作事件，下面将为读者列举一个程序实例。其程序实例代码如下所示：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test8
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JPopupMenu pop;
    JMenuItem item2;
    static JFrame f;
    JMenuItem item1;
    JPanel p;
    static JTabbedPane tp;
    public test8()
    {
        f=new JFrame("星火纺织厂信息系统");
        ...
        // 后面的代码内容与前面章节中的实例代码相似，这里不再列出
        buttonAction buttonaction=new buttonAction("磨砂分厂职工信息");
        // 创建按钮，并且引用动作类 buttonaction 和 menuaction
        button1.setAction(buttonaction);
        menuAction menuaction=new menuAction("退出");
        item4.setAction(menuaction);
    }
    public static void main(String args[])
    {
        new test8();
    }
}
class info extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    info()
    {
        ...
    }
}
```



```
// 与前面章节中相同实例的代码相似，这里不再列出
}
}
// 创建一个类，继承自 AbstractAction 类，执行动作事件
class buttonAction extends AbstractAction
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public buttonAction(String text)
    {
        super(text);
    }
    public void actionPerformed(ActionEvent arg0)
    {
        ...
        // 与前面章节中相同实例的代码相似，这里不再列出
    }
}
// 创建一个类，继承自 AbstractAction 类，执行动作事件
class menuAction extends AbstractAction
{
    public menuAction(String text)
    {
        super(text);
    }
    public void actionPerformed(ActionEvent arg0)
    {
        int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
            "退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            test.f.dispose();
        }
    }
}
```

上面程序代码的运行结果如图 19.9 所示。

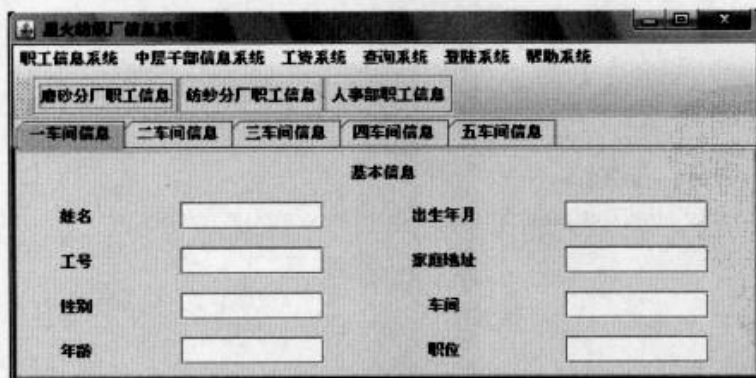


图 19.9 按钮和菜单的 Action 类

上面的程序将原先的添加监听器改为创建 Action 类，其实现与监听器是一样的。有兴趣的读者可以将前面章节中所有的事件监听器的实例改为使用 Action 类，看看结果运行如何。

19.5 如何支持辅助技术

什么是辅助技术呢？辅助技术也称无障碍技术，它使得残疾人士可以很方便地使用电脑。无障碍技术包括了语音接口、屏幕阅读器等一些输入设备，它不但可以适合残疾人士使用计算机，而且还可以在有些不方便使用计算机操作的情况下来操作计算机，例如可以通过语音输入和输出来查收邮件等。

无障碍技术是如何实现的呢？无障碍技术的工作原理是：在 API 中有一个接口称为 Accessible 接口，实现这个接口的对象就是无障碍对象，这个原理有点类似于线程的概念，实现 Runnable 接口的对象就是线程。实现这个接口需要实现其惟一的方法 getAccessibleContext()，该方法会返回一个 AccessibleContext 的中间对象，这个对象里面包含了无障碍对象的无障碍信息，所以创建无障碍对象只要实现 Accessible 接口中的 getAccessibleContext() 方法即可。

由于篇幅限制，所以不再对此技术进行详细讲解，希望有兴趣的读者可以查阅相关的资料。

19.6 如何使用焦点子系统

什么是焦点呢？不知道读者有没有过这样的经验：在软件的登录界面上有一个用户名的输入框和一个密码的输入框，当打开这个登录界面的时候，光标会自动停留在用户名的输入框中，当输入完密码时，按一下 Tab 键，光标又会停留在密码输入框中，等待用户输入。这个光标停留的位置就被称为焦点位置。

通常一个组件的焦点可以通过用户的鼠标单击获得，并且可以在组件与组件之间进行切换。当然本节介绍的是通过编程的方法获得焦点。下面将给出一个实例，此实例展现了窗口每次被激活时为特定组件设置焦点。其程序代码如下所示：

// 这段程序代码主要是为读者展示如何为组件指定默认焦点

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test9
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    static JButton button;
    public test9()
    {
        JFrame frame=new JFrame("焦点测试程序");
        JPanel pane=new JPanel();
        frame.setContentPane(pane);
        button=new JButton("测试按钮");
        pane.add(button);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
```



```
int y=(height-HEIGHT)/2;
frame.setLocation(x,y);
frame.setSize(WIDTH, HEIGHT);
frame.setVisible(true);
frame.pack();
frame.addWindowListener(new WindowAdapter()
{
    public void windowActivated(WindowEvent e)
    {
        button.requestFocusInWindow();
        // 让按钮组件在窗口被激活时, 立即获得动作焦点
    }
});
}
public static void main(String args[])
{
    new test9();
}
}
```

上面的程序运行结果是当 frame 窗口被激活时, button 组件会立即获得焦点。当然, 默认情况下会是这样, 下面再给出一个实例, 通过实例读者可以知道在多个组件中可以通过编程的方式设置任意组件的焦点。其程序代码如下所示:

// 这段程序代码主要展示如何在众多的组件中为组件指定默认焦点

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test10
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    static JButton button4;
    public test10()
    {
        JFrame frame=new JFrame("焦点测试程序");
        JPanel pane=new JPanel();
        frame.setContentPane(pane);
        JButton button1=new JButton("测试按钮一");
        JButton button2=new JButton("测试按钮二");
        JButton button3=new JButton("测试按钮三");
        button4=new JButton("测试按钮四");
        JButton button5=new JButton("测试按钮五");
        JButton button6=new JButton("测试按钮六");
        JButton button7=new JButton("测试按钮七");
        pane.add(button1);
        pane.add(button2);
        pane.add(button3);
        pane.add(button4);
        pane.add(button5);
        pane.add(button6);
        pane.add(button7);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
```



```
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
frame.setLocation(x,y);
frame.setSize(WIDTH, HEIGHT);
frame.setVisible(true);
frame.pack();
frame.addWindowListener(new WindowAdapter()
{
    public void windowActivated(WindowEvent e)
    {
        button4.requestFocusInWindow();
        // 让窗口被激活时, button4 按钮组件会自动获得焦点
    }
});
}
public static void main(String args[])
{
    new test10();
}
}
```

上面程序代码的运行结果如图 19.10 所示。



图 19.10 焦点子系统

上面的程序中设置了当激活窗口后,系统会自动将焦点设置到“测试按钮四”上。由此可以看出要设置一个组件为焦点最关键的就是使用 `requestFocusInWindow()` 方法。

在实际开发中的很多情况下都需要自定义组件的焦点。要想让一个组件获得焦点,此组件必须要满足三个要求,即可见性、被启用、可获得焦点。有的组件在默认情况下不一定能够获得焦点,此时就必须使用 `setFocusable()` 方法使得该组件可以获得焦点。

下面将讲述什么是焦点子系统。在一个框架容器中,可能包括了很多组件,每个组件都有自己的焦点,整个框架容器中所有组件的焦点就形成了一个焦点子系统。正是因为此焦点子系统中拥有很多个焦点,所以就涉及到了一个焦点移动的顺序,使用专业词汇来说就是焦点遍历顺序。常用的焦点子系统遍历键是 Tab 键。一个 Swing 开发的程序的焦点遍历策略是由 `LayoutFocusTraversalPolicy` 类来确定的。在实际开发中,也可以自定义 `FocusTraversalPolicy` 类来设置自己的焦点遍历策略。

下面给出一个实例,通过该实例,读者可以清楚地了解如何使用遍历策略来规范容器中的各个焦点的访问顺序。其程序代码如下所示:

```
// 这段程序代码主要为读者展示如何使用遍历策略来规范众多组件的焦点子系统
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class test11
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
```



```
static JButton button1;
static JButton button2;
static JButton button3;
static JButton button4;
static JButton button5;
static JButton button6;
static JButton button7;
public test11()
{
    JFrame frame=new JFrame("焦点测试程序");
    JPanel pane=new JPanel();
    frame.setContentPane(pane);
    button1=new JButton("测试按钮一");
    button2=new JButton("测试按钮二");
    button3=new JButton("测试按钮三");
    button4=new JButton("测试按钮四");
    button5=new JButton("测试按钮五");
    button6=new JButton("测试按钮六");
    button7=new JButton("测试按钮七");
    pane.add(button1);
    pane.add(button2);
    pane.add(button3);
    pane.add(button4);
    pane.add(button5);
    pane.add(button6);
    pane.add(button7);
    Toolkit kit=Toolkit.getDefaultToolkit();
    Dimension screenSize=kit.getScreenSize();
    int width=screenSize.width;
    int height=screenSize.height;
    int x=(width-WIDTH)/2;
    int y=(height-HEIGHT)/2;
    frame.setLocation(x,y);
    frame.setSize(WIDTH, HEIGHT);
    frame.setVisible(true);
    frame.pack();
    framefocus focus1=new framefocus();
    frame.setFocusTraversalPolicy(focus1);
    frame.addWindowListener(new WindowAdapter()
    {
        public void windowActivated(WindowEvent e)
        {
            button1.requestFocusInWindow();
        }
    });
}
public static void main(String args[])
{
    new test11();
}
class framefocus extends FocusTraversalPolicy
{
    // 焦点遍历策略, 这里是向后遍历, 也就是说当按 Tab 键时, 当前按钮组件的下一个按钮组件是哪一个
    public Component getComponentAfter(Container arg0, Component arg1)
    {
        if(arg1.equals(test.button1)) // 如果当前组件是按钮一的话, 则下一个遍历对象则是按钮二
```



```
{
    return test.button2;
}
else if(arg1.equals(test.button2))
{
    return test.button4;
}
else if(arg1.equals(test.button4))
{
    return test.button3;
}
else if(arg1.equals(test.button3))
{
    return test.button5;
}
else if(arg1.equals(test.button5))
{
    return test.button7;
}
else if(arg1.equals(test.button7))
{
    return test.button6;
}
return test.button1;
}

public Component getComponentBefore(Container arg0, Component arg1)
{
    // 焦点遍历策略, 这里是向前遍历, 也就是说当按 Tab 键时, 当前按钮组件的下一个按钮组件是哪一个
    if(arg1.equals(test.button6))
    {
        return test.button7;
    }
    else if(arg1.equals(test.button7))
    {
        return test.button5;
    }
    else if(arg1.equals(test.button5))
    {
        return test.button3;
    }
    else if(arg1.equals(test.button3))
    {
        return test.button4;
    }
    else if(arg1.equals(test.button4))
    {
        return test.button2;
    }
    else if(arg1.equals(test.button2))
    {
        return test.button1;
    }
    return test.button1;
}

public Component getDefaultComponent(Container arg0)
{
    return test.button1;
}
```



```
public Component getFirstComponent(Container arg0)
{
    return test.button1;
}
public Component getLastComponent(Container arg0)
{
    return test.button6;
}
}
```

上面程序代码的运行结果如图 19.11 所示。



图 19.11 焦点子系统的遍历

上面的程序自定义了焦点遍历策略，其遍历顺序是“测试按钮一”→“测试按钮二”→“测试按钮四”→“测试按钮三”→“测试按钮五”→“测试按钮七”→“测试按钮六”。当按 Tab 键时，系统就会按照以上自定义的顺序设置组件的焦点。

有关焦点子系统的知识还有很多，希望读者自行查阅相关资料多多练习，并且将前面讲过的知识融合在一起，设计出具有个人特色的软件程序。

19.7 如何使用键绑定

本节将讲述有关键绑定的知识。在讲解菜单的时候曾涉及到了快捷键的知识，本节所讲述的键绑定就很类似于快捷键，可通过它操作键盘来达到代替鼠标操作软件的目的。

对键绑定支持是由 JComponent 提供的，并且依赖 InputMap(输入映射表)类和 ActionMap(动作映射表)类。输入映射表是将键盘的敲击与动作名称绑定，而动作映射表是为每一个动作名称指定响应的动作。然而，每一个 JComponent 都会有一个动作映射表和三个输入映射表，这三个输入映射表分别对应三种不同的输入映射表情况，如表 19.3 所示。

表 19.3 输入映射表

输入映射表	说明
JComponent.WHEN_FOCUSED	组件具有键盘焦点。在组件没有子组件的情况下可使用此表，例如按钮、文本输入框等
JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT	这个输入映射表是组合组件中最常用的——组合组件是指其实现依赖于子组件的组件，例如表格中使用它作为所有绑定的输入映射表
JComponent.WHEN_IN_FOCUSED_WINDOW	它通常用于快捷键。在这里需要强调的是当用户按下一个键时，JComponent 键事件处理代码将搜索这些输入映射表，找到这个键有效的绑定，接着在动作映射表中查询响应的动作

创建一个组件 JComponent 的键绑定的方法如下：

- JComponent.getInputMap().put(KeyStroke.getKeyStroke("键"), "dosomething")用于创建一个输入映射表项。
- JComponent.getActionMap().put("dosomething", action)用于创建一个动作映射表中的动作项。

下面将给出一个实例，通过这个实例可使读者能够清楚了解如何使用键绑定，其程序代码如下所示：

```
// 这段程序代码主要为读者展示如何为组件指定键绑定功能
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class test12
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JPopupMenu pop;
    JMenuItem item2;
    static JFrame f;
    JMenuItem item1;
    JPanel p;
    static JTabbedPane tp;
    public test12()
    {
        ...
        // 与前面章节中相同实例的代码相似，这里不再列出
        buttonAction buttonaction=new buttonAction("磨砂分厂职工信息");
        button1.setAction(buttonaction);
        menuAction menuaction=new menuAction("退出");
        item4.setAction(menuaction);
        // 将快捷键 B 同 button1 绑定，也就是说当按 B 时就相当于单击 button1 按钮组件
        button1.getInputMap(2).put(KeyStroke.getKeyStroke("B"), "open");
        button1.getActionMap().put("open", buttonaction);
        item4.getInputMap(2).put(KeyStroke.getKeyStroke("A"), "QUIT");
        item4.getActionMap().put("QUIT", menuaction);
    }
    public static void main(String args[])
    {
        new test12();
    }
}
class info extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    info()
    {
        ...
        // 与前面章节中相同实例的代码相似，这里不再列出
    }
}
```



```
class buttonAction extends AbstractAction
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public buttonAction(String text)
    {
        super(text);
    }
    public void actionPerformed(ActionEvent arg0)
    {
        ...
        // 与前面章节中相同实例的代码相似，这里不再列出
    }
}

class menuAction extends AbstractAction
{
    public menuAction(String text)
    {
        super(text);
    }
    public void actionPerformed(ActionEvent arg0)
    {
        int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
            "退出确认对话框", JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            test.f.dispose();
        }
    }
}
```

上面的程序运行后，当单击键盘 B 键后，其作用同 button1 按钮作用相同，当单击键 A 后，会与 item4 菜单项作用相同，这样就形成了键绑定的效果。

19.8 如何在对话框中使用 Modality

Modality 翻译过来就是指模态窗口，用来实现模态对话框，所谓模态对话框，就是具有这样特性的对话框：除非把它关掉，否则无法切换到它的父窗口上。本节将会详细地为读者讲述有关模态对话框的知识。

Java 6 支持如下 4 种模态类型。

- 无模式：所谓的无模式也就是说对话框在自己可见时并不阻断任何其他窗口。
- 文档模式：所谓的文档模式也就是说对话框阻断所有来自于同一个文档的窗口，除了那些来自于它的子层次上的窗口外。
- 应用程序模式：所谓的应用程序模式也就是说对话框能够阻断同一应用程序中的所有窗口，除了那些来自它的子层次上的窗口外。
- 工具箱模式：所谓的工具箱模式也就是说对话框能阻断所有运行于同样的工具箱中的窗口，除了那些来自于它的子层次上的窗口外。

这 4 种模态形式在实际开发应用中是按照其模态优先权来进行处理的,也就是说优先权高的模式会覆盖优先权低的模式。其优先权根据其阻断强度来排序:无模式、文档模式、应用程序模式、工具箱模式,如图 19.12 所示。

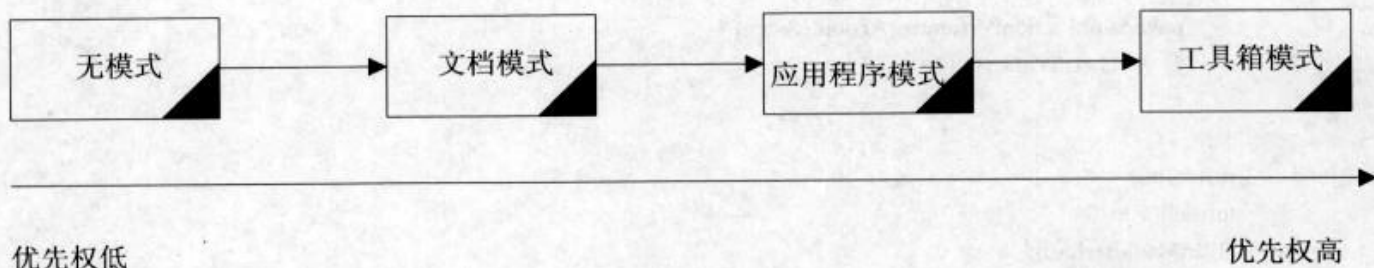


图 19.12 模态优先权示意图

下面为了能够让读者更加清晰地理解上面几个概念,给出一个程序实例,其实例代码如下所示:

```
// 这段程序代码主要是为读者展示如何使用不同的模态形式创建顶层窗口
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class test13
{
    private static Frame f1;
    private static Dialog d11;
    private static Dialog d12;
    private static Frame f2;
    private static Dialog d21;
    private static Dialog d22;
    private static JFrame f3;
    private static JFrame f4;
    private static FileDialog fd4;
    private static WindowListener closeWindow = new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            e.getWindow().dispose();
        }
    };
    public static void main(String[] args)
    {
        GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice gd = ge.getDefaultScreenDevice();
        GraphicsConfiguration gc = gd.getDefaultConfiguration();
        int sw = gc.getBounds().width - 64;
        int sh = gc.getBounds().height - 64;
        JLabel l;
        JButton b;
        MenuBar mb;
        Menu m;
        MenuItem mi;
        Font labelFont = new Font("Tahoma", 24, Font.PLAIN);
        // 第 1 个顶层容器
        f1 = new Frame("Parent Frame");
        f1.setBounds(32, 32, 300, 200);
        f1.addWindowListener(closeWindow);
```



```
// 创建菜单
mb = new MenuBar();
m = new Menu("Test");
mi = new MenuItem("Show modeless");
mi.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        d11.setVisible(true);
    }
});
m.add(mi);
mb.add(m);
f1.setMenuBar(mb);
f1.setLayout(new BorderLayout());
l = new Label("FRAME");
l.setAlignment(Label.CENTER);
l.setFont(labelFont);
l.setBackground(Color.RED);
f1.add(l, BorderLayout.CENTER);
f1.setVisible(true);
d11 = new Dialog(f1, "Modeless Dialog");
d11.setBounds(132, 132, 300, 200);
d11.addWindowListener(closeWindow);
d11.setLayout(new BorderLayout());
l = new Label("MODELESS");
l.setAlignment(Label.CENTER);
l.setFont(labelFont);
l.setBackground(Color.RED);
d11.add(l, BorderLayout.CENTER);
b = new Button("Show document-modal");
b.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        d12.setVisible(true);
    }
});
d11.add(b, BorderLayout.SOUTH);
d12 = new Dialog((Window)d11, "Document-modal Dialog", Dialog.ModalityType.DOCUMENT_MODAL);
// 创建对话框
d12.setBounds(232, 232, 300, 200);
d12.addWindowListener(closeWindow);
d12.setLayout(new BorderLayout());
l = new Label("DOCUMENT_MODAL");
l.setAlignment(Label.CENTER);
l.setFont(labelFont);
l.setBackground(Color.RED);
d12.add(l, BorderLayout.CENTER);
f2 = new Frame("Parent Frame");
f2.setBounds(sw - 300 + 32, 32, 300, 200);
f2.addWindowListener(closeWindow);
mb = new MenuBar();
m = new Menu("Test");
mi = new MenuItem("Show modeless");
mi.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        d21.setVisible(true);
    }
});
```

// 设置排列方式

// 设置边框颜色

// 创建一个新对话框

// 设置边框

// 设置边框颜色

// 设置边框

// 设置边框颜色


```

    });
    m.add(mi);
    mb.add(m);
    f2.setMenuBar(mb);
    f2.setLayout(new BorderLayout());
    l = new Label("FRAME");
    l.setBackground(Color.BLUE);
    l.setAlignment(Label.CENTER);
    l.setFont(labelFont);
    f2.add(l, BorderLayout.CENTER);
    f2.setVisible(true);
    d21 = new Dialog(f2, "Modeless Dialog");           // 创建新对话框
    d21.setBounds(sw - 400 + 32, 132, 300, 200);       // 设置其边框
    d21.addWindowListener(closeWindow);
    d21.setLayout(new BorderLayout());
    l = new Label("MODELESS");
    l.setBackground(Color.BLUE);                       // 设置边框颜色
    l.setAlignment(Label.CENTER);
    l.setFont(labelFont);
    d21.add(l, BorderLayout.CENTER);
    b = new Button("Show document-modal");
    b.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            d22.setVisible(true);
        }
    });
    d21.add(b, BorderLayout.SOUTH);
    d22 = new Dialog((Window)d21, "Document-modal Dialog", Dialog.ModalityType.DOCUMENT_MODAL);
}

```

上面程序代码的运行结果如图 19.13 所示。

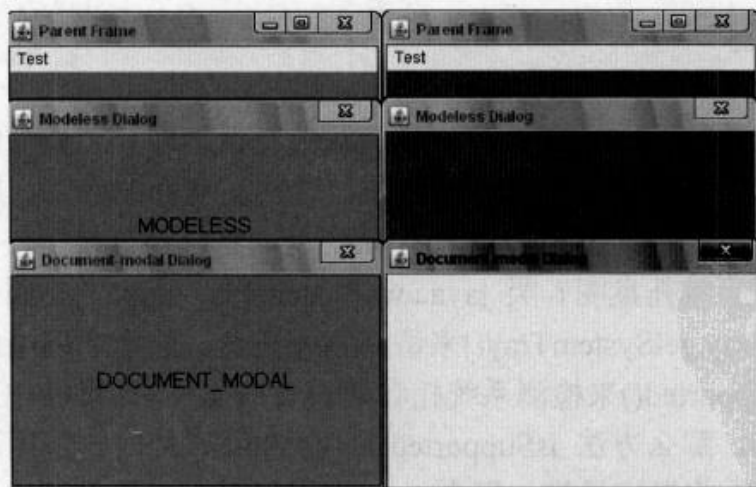


图 19.13 模态对话框

在上例中，当没有关闭文档模式的对话框时，是无法关闭其上层窗口的。而无模式则无所谓，可以在不关闭自己的前提下关闭父窗口。通过这个实例，读者可以清楚地看到模态对话框的几种类型的不同点。

至于模态对话框的相关知识还有很多，这里就不再赘述了，希望读者能够查阅相关的资料自行举例练习。

19.9 如何创建 Splash Screen

什么是 Splash Screen 呢？在 Java 虚拟机启动之前，可以在应用程序启动时创建闪现屏幕，而这个闪现的屏幕可显示为一个包含图像的未装饰的窗口，这个闪现的屏幕就被称为 Splash Screen。当然这个图像可以使用各种格式的文件，例如 GIF（支持动画）、JPEG、PNG 等。当程序的第一个界面出现时，这个 Splash Screen 就会自动消失。

如何才能出现 Splash Screen，Splash Screen 类可用于关闭闪现屏幕、更改闪现屏幕图像、获取图像的位置和大小以及在闪现屏幕中进行绘制。它不能用于创建闪现屏幕，应当使用命令行或清单文件选项来实现该操作。在实际应用中，显示或者创建本机闪现屏幕有两种方式。

如果是用命令行或快捷方式运行应用程序，则使用“-splash:“Java”应用程序启动器选项来显示闪现屏幕。例如：

```
java -splash:filename.gif Test
```

如果应用程序被打包在“.jar”文件中，可以使用清单文件中的 SplashScreen-Image 选项来显示闪现屏幕。将图像放在“.jar”归档文件中并用选项指定路径。路径不应以斜杠开头。例如，在 manifest.mf 文件中：

```
Manifest-Version: 1.0  
Main-Class:Test  
SplashScreen-Image:filename.gif
```

然而，命令行接口的优先级高于清单设置。另外有一点需要强调的是：Splash Screen 类是不可以实例化的，要想获得其对象，必须通过 getSplashScreen()方法来获得。如果应用程序既不以命令行也不以清单文件的形式创建启动画面的话，那么 getSplashScreen()方法就会返回一个 null，有兴趣的读者可以自行查阅相关资料并举例练习，本书不再针对此详述了。

19.10 如何使用 System Tray

什么是 System Tray 呢？System Tray 又称为系统托盘，即一个软件的最小化图标，但是与最小化图标又有不同：系统托盘是个特殊区域，通常在桌面的底部，用户可以随时访问正在运行中的程序。在平时使用的 Windows 操作系统里，系统托盘常指任务栏的状态区域。

那么如何才能使用系统托盘呢？类 java.awt.SystemTray 代表了桌面上的系统托盘。通过调用静态方法 SystemTray.getSystemTray()来访问系统托盘。然而在调用此方法前，应该用静态方法 SystemTray.isSupported()来检测系统托盘是否可被支持。如果操作系统上系统托盘还未准备就绪或者不被支持，那么方法 isSupported()返回 false。此时，应用程序如果试图调用方法 getSystemTray()，那么将抛出异常，每个 Java 应用程序有一个单一的 SystemTray 实例。

一个应用程序不应创建它本身的 SystemTray 实例，而是应通过调用方法 getSystemTray()来获得当前可用的系统托盘。系统托盘包含一个或多个托盘图标，这些图标是以调用方法 add(java.awt.TrayIcon)添加的，当不需要这些图标时，通过调用方法 remove(java.awt.TrayIcon)来删除。

最小化到状态栏中的系统托盘是一个图标，那么这个图标该如何创建呢？系统托盘允许添加或者删除一个以及多个 java.awt.TrayIcon 实例。TrayIcon 对象实例代表了可被添加进系统托盘的图标。但是，TrayIcon 的功能可不仅仅是在托盘上显示图标那么简单，它可以具有工

具提示文本、一个 AWT 的弹出式菜单以及一系列监听器。

下面给出一个简单实例，可使读者能够清楚如何使用系统托盘。其程序代码如下所示：

```
// 这段程序代码主要是为读者展示如何使用系统托盘
import javax.swing.*;
import java.awt.*;
public class test14
{
    public static void main(String[] args)
    {
        if(SystemTray.isSupported())
        {
            SystemTray tray = SystemTray.getSystemTray();
            Image image = Toolkit.getDefaultToolkit().getImage("d:/btn120-spa.gif");
            PopupMenu popup = new PopupMenu();
            MenuItem item = new MenuItem("弹出菜单");
            popup.add(item);
            TrayIcon trayIcon = new TrayIcon(image, "系统托盘信息", popup);
            try
            {
                tray.add(trayIcon);
            }
            catch (Exception e)
            {
                System.err.println("无法向这个托盘添加新项: " + e);
            }
        }
        else
        {
            System.err.println("无法使用系统托盘!");
        }
    }
}
```

上面程序代码的运行结果如图 19.14 所示。



图 19.14 系统托盘的使用

图中最左边的图标就是上面程序在系统托盘中添加的图标。从上面的程序可以总结出使用系统托盘的步骤。

- 01 首先获得一个系统托盘的对象：SystemTray tray = SystemTray.getSystemTray()。
- 02 获得一个图标的资源：Image image = Toolkit.getDefaultToolkit().getImage("d:/btn120-spa.gif")。
- 03 创建一个图标对象：TrayIcon trayIcon = new TrayIcon(image, "系统托盘信息", popup)。
- 04 将图标对象添加到系统托盘中：tray.add(trayIcon)。

当然，系统托盘还有很多相关知识，但已经超出了本书的范围，希望读者能够将以前学习过的知识与本节的知识进行综合练习，以巩固整个系统知识。

19.11 如何使用 Swing 拖曳功能和数据传输

说到拖曳功能和数据传输，大家应该不会陌生，在 Windows 操作系统的资源管理器中，

当从一个文件夹中选中的一个文件或者一批文件，并拖曳到另一个文件夹中去，这就是一个简单的拖曳功能实现的数据传输的实例。

要想实现 Swing 拖曳功能和数据传输总共要分三个步骤：

- 01 必须要实现一个拖曳源，拖曳源和相应的支持拖曳功能的组件是关联起来的。
- 02 必须要实现一个拖曳目标，这个目标用来实现拖曳物的接收。
- 03 必须实现一个数据传输对象，该对象用于封装拖动数据。

下面给出一个示意图，使读者能够清楚如何实现此功能，如图 19.15 所示。

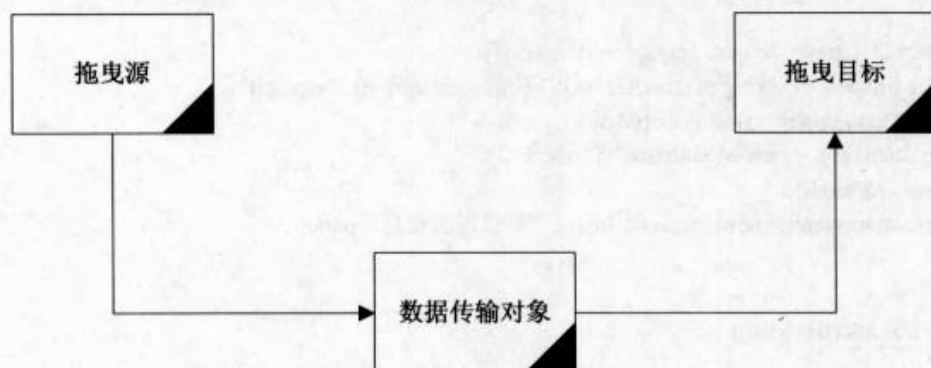


图 19.15 拖曳功能和数据传输示意图

数据传输机制的核心就是 `TransferHandler` 接口，它提供了组件之间传输数据的简单机制。要传输的数据被绑定在一个实现了 `TransferHandler` 接口的类上。可以通过 `setTransferHandler()` 方法来设置传输句柄，这是最简单的拖曳功能和数据传输。

下面将给出实例，通过实例可使读者能够真正了解利用拖曳功能实现数据传输的方法。其程序实例代码如下所示：

```
// 这段程序代码主要展示如何从颜色选择器中将颜色信息拖曳到其他组件中，使得目的组件中的颜色发生变化
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class test15
{
    private JFrame mainFrame;
    private JPanel mainPanel;
    private JLabel label;
    private JTextField textField;
    private JColorChooser colorChooser;
    private JMenuBar menuBar = new JMenuBar();
    private JMenu menu = new JMenu("Menu");
    private JMenuItem menuItem = new JMenuItem("Handle Foreground");
    private TransferHandler t1 = new TransferHandler("text");
    private TransferHandler t2 = new TransferHandler("foreground");
    public test15()
    {
        mainFrame = new JFrame();
        mainPanel = new JPanel(new BorderLayout());
        label = new JLabel("label");
        label.setTransferHandler(t1);
        menuItem.addActionListener(new ActionListener()
        {
```



```
public void actionPerformed( ActionEvent e )
{
    if( label.getTransferHandler().equals( t1 ) )
    {
        test.this.menuitem.setText( "Handle Text" );
        label.setTransferHandler( t2 );
    }
    else
    {
        test.this.menuitem.setText( "Handle Foreground" );
        label.setTransferHandler( t1 );
    }
}

});
menu.add( menuitem );
menu.setTransferHandler( t1 );
menuBar.add( menu );
mainFrame.setJMenuBar( menuBar );
label.addMouseListener( new MouseAdapter()
{
    public void mousePressed( MouseEvent e )
    {
        // 设置拖动源, 获得拖动目标, 设置数据传输
        JComponent c = (JComponent)e.getSource();
        TransferHandler handler = c.getTransferHandler();
        handler.exportAsDrag( c, e, TransferHandler.COPY );
    }
}
});
textField = new JTextField( 20 );
textField.setDragEnabled( true );
colorChooser = new JColorChooser();
colorChooser.setDragEnabled( true );
mainPanel.add( label, BorderLayout.PAGE_START );
mainPanel.add( textField, BorderLayout.PAGE_END );
mainPanel.add( colorChooser, BorderLayout.WEST );
mainFrame.getContentPane().add( mainPanel );
mainFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
mainFrame.pack();
mainFrame.setLocationRelativeTo( null );
mainFrame.setVisible( true );
}

public static void main( String[] args )
{
    new test15();
}
}
```

上面程序的运行结果如图 19.16 所示。

pin51.com
最新编程资源分享
爱拼才会赢

PDF

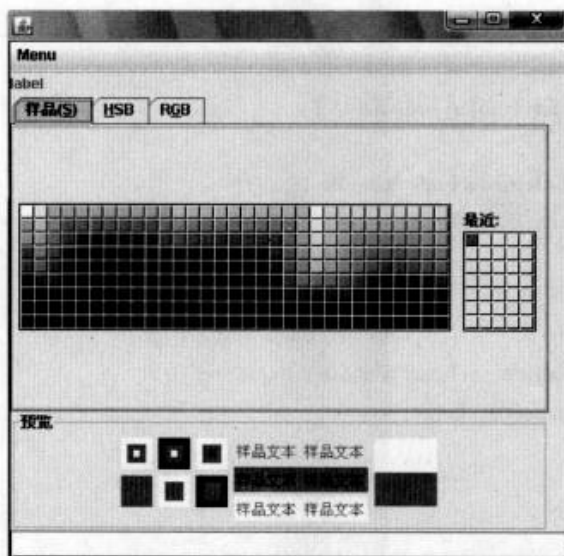


图 19.16 拖曳功能与数据传输

当在菜单中选择并从预览中选择颜色拖到 label 上, label 的颜色就与预览中的颜色一样了, 从而实现了拖曳功能和数据传输功能。有关这方面的知识, 还有很多内容没有为读者讲述, 限于篇幅, 这里不再详述了, 希望大家能够自行学习。

19.12 本章小结

本章主要讲述了一些实际开发中所需要的零碎技巧知识, 这些知识不一定是开发中必备的, 但是它是一个软件是否能够具有专业性的标志。

19.13 本章习题

1. 设计一个程序, 程序中有一个菜单, 请为菜单中所有的菜单项设置快捷键, 菜单中有三个菜单项, 分别是“打开第一个窗口”、“打开第二个窗口”、“关闭第一、二个窗口”。其快捷键分别为 G、H、F, 使得菜单中所有的菜单项都是利用键盘控制的。

要求: 其最终结果如图 19.17 所示。

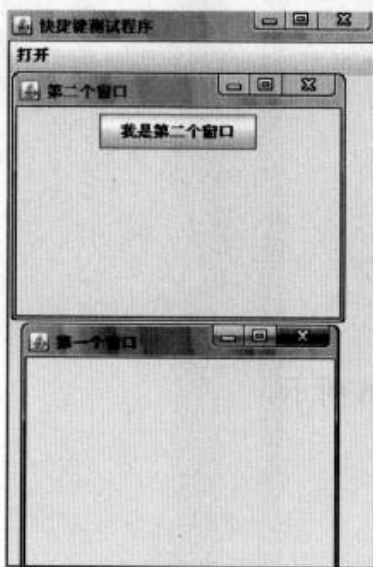


图 19.17 本章习题 1

2. 设计一个菜单，在菜单中有几个菜单项，用来改变顶层容器的边框颜色。
3. 设计一个程序，在程序界面上有三个按钮组件、两个文本组件，使用程序将其遍历的顺序变为：“按钮一”→“文本组件一”→“按钮二”→“文本组件二”→“按钮三”。

要求：

- (1) 其运行后的最终结果如图 19.18 所示。

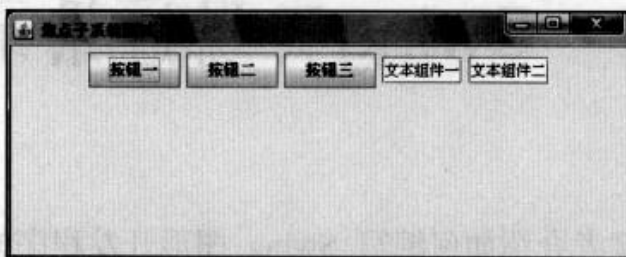


图 19.18 本章习题 3

- (2) 当按 Tab 键时，会按照题目中所给的顺序依次获得焦点。

4. 设计一个简单的登录窗口，当开始运行程序时，首先焦点会停留在“姓名”文本框上，当按 Tab 键时，会将焦点转移到“密码”文本框上。

要求：

- (1) 其运行的结果如图 19.19 所示。

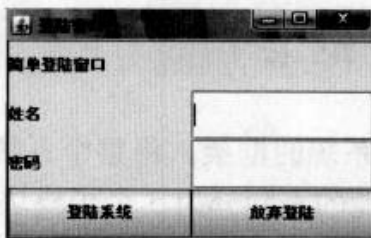


图 19.19 本章习题 4

- (2) 程序会按照“姓名”文本框→“密码”文本框→“登录系统”按钮→“放弃登录”按钮的遍历顺序来设置焦点子系统。

第20章 Swing实现通讯录系统

本章的主要目的是为读者介绍如何编写 Swing 图形开发程序的一个实例——通讯录系统。为了能够让读者真正地学会如何利用面向对象编程的方法编写 Java 程序，本章将整个程序分成若干个类，而每一个类就是一个小系统，并且针对每个小系统，为读者详细讲述如何编写其程序代码。最后将为读者讲述如何将这些小系统整合成通讯录系统。在系统中将会使用到很多组件，包括按钮组件、标签组件、框架组件、容器组件、树组件、菜单组件、分隔窗口容器组件等，希望读者能够紧紧跟随作者的思路，仔细阅读本章，从而能够学习到编程的方法与实质。

20.1 通讯录系统的软件框架

本节将为读者分析整个通讯录系统的框架，将整个系统分成两个大模块：登录系统和主菜单系统。而主菜单系统又按照功能的不同分为 5 个子模块：同学通讯系统、同事通讯系统、朋友通讯系统、查询系统、帮助系统。

- 同学通讯系统包括两个模块，即同学基本系统模块和同学联系方式模块。
- 同事通讯系统包括同事基本信息模块和同事联系方式模块。
- 朋友通讯系统包括朋友基本信息模块和朋友联系方式模块。
- 查询系统包括同事查询系统、同学查询系统和朋友查询系统。
- 帮助系统包括版本信息系统和帮助文档系统。

整个软件界面的框架如图 20.1 所示。

以上是整个软件系统的界面框架，但是一个软件仅有界面是无用的，必须要有数据，所以，整个软件界面必须要以数据库为底层。

那么如何操作数据库呢？如果直接操作数据库的话，会出现一个问题。假设是网络数据库，那么每次要提取数据都必须通过网络来操作数据库，这样做无疑是很不方便的，一旦网络故障将会影响数据库的操作，或者说一旦通过网络操作数据库，也无疑会占用很多带宽资源。为了避免这种情况的发生，会先将数据库中的数据存储到本地的 Vector 的数据结构中，然后直接操作这个 Vector 数据结构，就相当于操作数据库。一旦操作完 Vector 中的数据后再提交到数据库中，从而加快速度。

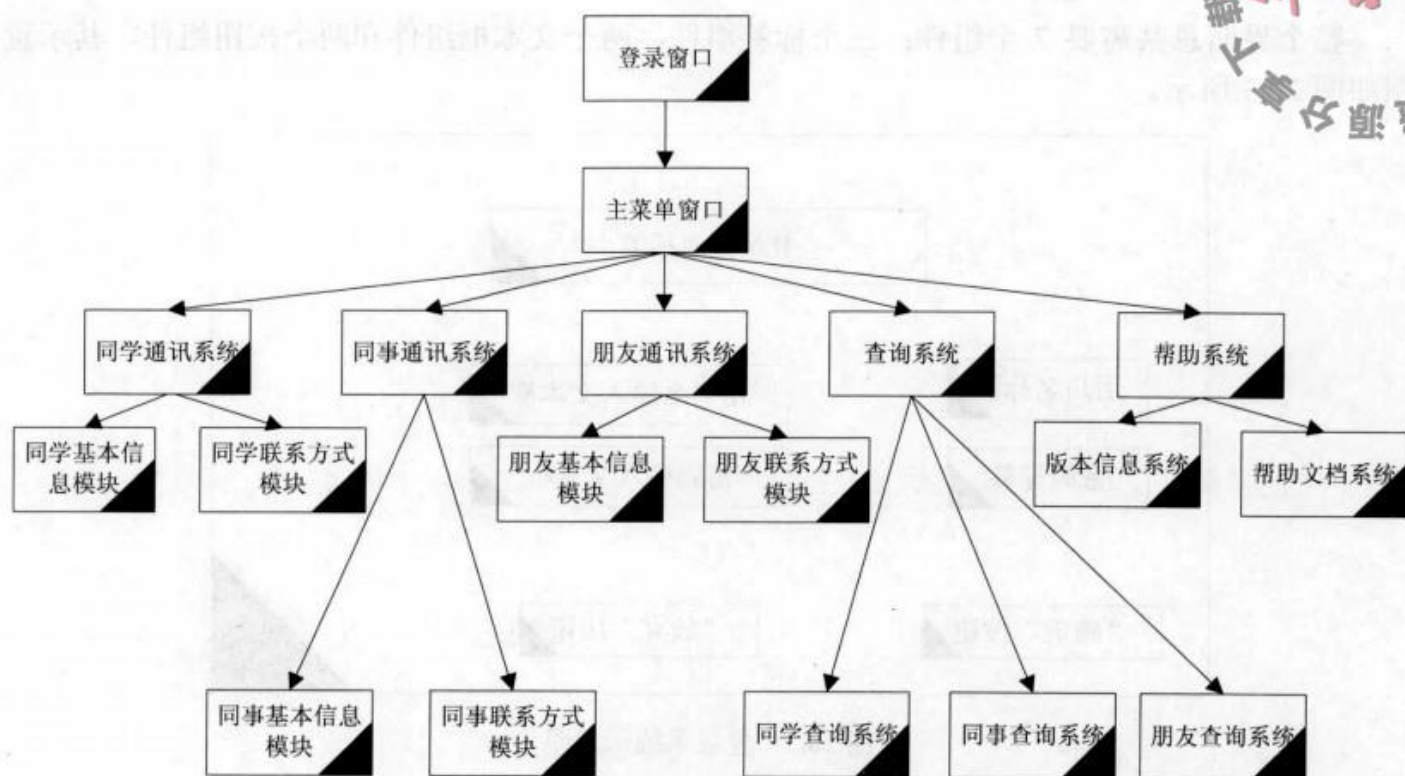


图 20.1 通讯录系统界面框架示意图

下面将通过示意图让读者更加清楚这种操纵数据库的方式，如图 20.2 所示。

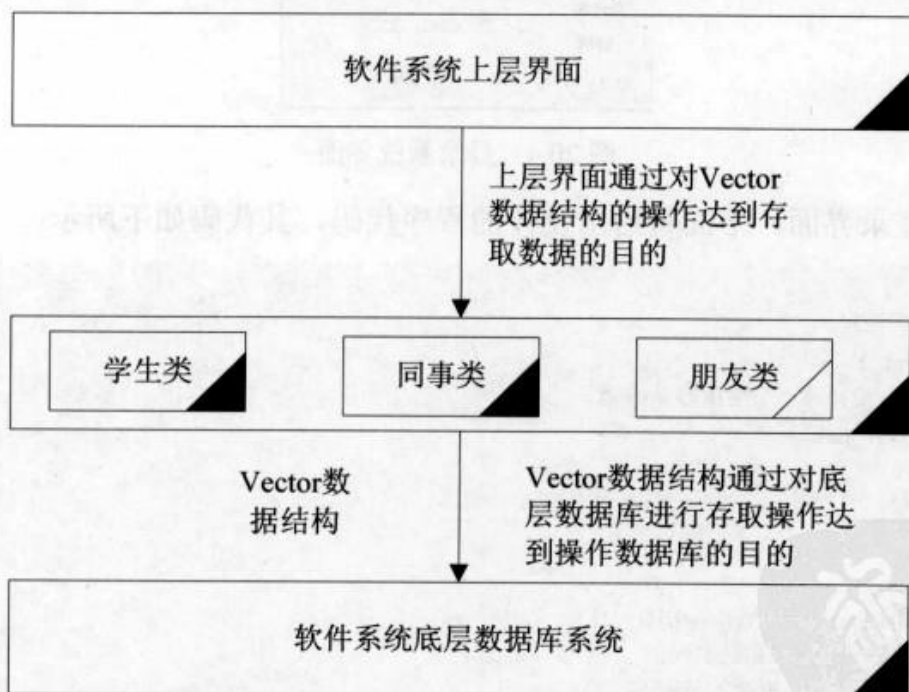


图 20.2 如何操作底层数据库示意图

下面将为读者讲解各个模块，同时会配上带详细注解的程序实例代码。

20.2 通讯录系统的登录系统

在日常的生活中，例如银行 ATM 机上的登录系统、Windows 操作系统的登录系统等，读者一定不会陌生。其实，登录系统的主要作用是让授权的用户能够访问系统，所以登录系统一般会包括对用户名和密码的输入和验证。本节将为读者讲述如何设计通讯录系统的登录界面。

整个界面总共需要 7 个组件：三个标签组件、两个文本框组件和两个按钮组件。其示意图如图 20.3 所示。

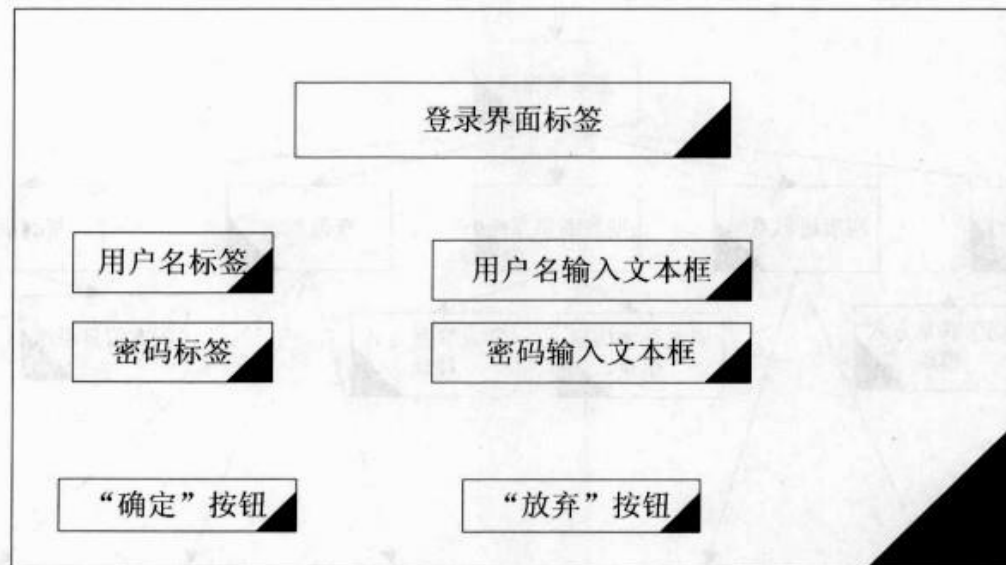


图 20.3 登录系统示意图

针对上面的示意图，设计出如图 20.4 所示的登录系统界面。

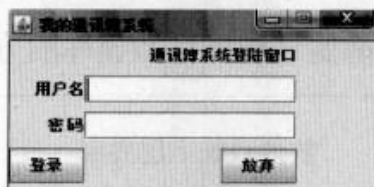


图 20.4 登录系统界面

针对上图的登录界面，下面将给出设计的程序代码，其代码如下所示：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
// 这是一个登录类。设计成一个继承容器的类
class loggins extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    // WIDTH 是指整个顶层框架的宽度
    // HEIGHT 是指整个顶层框架的长度
    static final int WIDTH=300;
    static final int HEIGHT=150;
    JFrame loginframe;
    // 此方法用来添加控件到容器中
    // 按照网格组布局方式排列组件的方法
    // x 指控件位于第几列
    // y 指控件位于第几行
    // w 指控件需要占几列
    // h 指控件需要占几行
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
```



```

constraints.gridy=y;
constraints.gridwidth=w;
constraints.gridheight=h;
add(c,constraints);
}
// 这是一个构造器方法
// loginframe 指这个界面的框架
// setDefaultCloseOperation 是一个使得窗口上面的关闭控件有效的类库方法
// lay 是一个网格组布局管理器的对象
// nameinput 是用来输入用户名的文本域
// passwordinput 是用来输入密码的文本域
// title 是用来显示标题的标签
// name 是用来显示“姓名”的标签
// password 是用来显示“密码”的标签
// OK 是一个按钮, 用于进入系统
// cancel 是一个按钮, 用于退出界面和系统
// ok.addActionListener 是一个进入系统动作事件的监听方法
// cancel.addActionListener 是一个退出系统和界面动作事件的监听方法
logins()
{
    loginframe=new JFrame("我的通讯簿系统");
    loginframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    GridBagLayout lay=new GridBagLayout();
    setLayout(lay);
    loginframe.add(this, BorderLayout.WEST);
    loginframe.setSize(WIDTH,HEIGHT);
    Toolkit kit=Toolkit.getDefaultToolkit();
    Dimension screenSize=kit.getScreenSize();
    int width=screenSize.width;
    int height=screenSize.height;
    int x=(width-WIDTH)/2;
    int y=(height-HEIGHT)/2;
    loginframe.setLocation(x,y);
    JButton ok=new JButton("登录");
    JButton cancel=new JButton("放弃");
    JLabel title=new JLabel("通讯簿系统登录窗口");
    JLabel name=new JLabel("用户名");
    JLabel password=new JLabel("密 码");
    final JTextField nameinput=new JTextField(15);
    final JTextField passwordinput=new JTextField(15);
    GridBagConstraints constraints=new GridBagConstraints();
    constraints.fill=GridBagConstraints.NONE;
    constraints.anchor=GridBagConstraints.EAST;
    constraints.weightx=4;
    constraints.weighty=4;
    add(title,constraints,2,0,2,1);
    add(name,constraints,1,1,2,1);
    add(password,constraints,1,2,2,1);
    add(nameinput,constraints,3,1,2,1);
    add(passwordinput,constraints,3,2,2,1);
    add(ok,constraints,1,3,1,1);
    add(cancel,constraints,3,3,1,1);
    loginframe.setResizable(false);
    loginframe.setVisible(true);
    ok.addActionListener(new ActionListener()
    { // 当单击“确定”按钮后, 会引起动作事件, 下面的代码将给出事件处理的结果

```

// 通过下面的代码来设置登录窗口的位置

// 设置网格组布局管理器的参数

// 使用网格组布局添加控件


```
public void actionPerformed(ActionEvent Event)
{
    String nametext=nameinput.getText();
    String passwordtext=passwordinput.getText();
    String str=new String(passwordtext);
    boolean x=(nametext.equals("starsong"));
    boolean y=(str.equals("750720"));           // 在此设置密码和用户名
    boolean z=(x&& y);
    if (z==true)
    { // 当密码和用户名都正确的话, 将会进入到主菜单界面
        // 如果用户名或者密码不正确, 那么系统会将所有输入清空, 让用户重新输入
        new mainframe();                       // 此界面就是后面的主界面
        loginframe.dispose();
    }
    else if(z==false)
    {
        nameinput.setText("");
        passwordinput.setText("");
    }
}
});
cancel.addActionListener(new ActionListener()
{ // 当单击“放弃”按钮后, 会引起动作事件, 下面的代码给出了此事件处理结果的方式
    public void actionPerformed(ActionEvent Event)
    {
        loginframe.dispose();
    }
});
}
}
public class login
{
    public static void main(String[] args)
    {
        new loggins();
    }
}
```

当输入正确的用户名和密码后, 就会进入到主菜单界面。如果输入错误的话, 系统将会清除所有的输入, 让用户重新输入。当然, 还可以将登录的条件更加复杂化, 其关键在于如何针对按钮控件的事件进行适当的处理。登录系统设计的关键在于如何处理两个按钮控件的事件处理。

20.3 通讯录系统的主菜单系统

本节将讲述主菜单界面的设计。为了能够让读者清楚整个设计的过程, 下面将主菜单的几个模块分别给予详细讲述。其模块分为: 数据库模块、信息界面模块、功能模块、其他模块。

其中数据库模块将为读者讲述如何建立数据库、如何建立数据源、基本类的设计、如何将数据库中的数据存储到 Vector 数据结构中去。而信息界面模块将主要为读者讲述如何针对同学通讯系统、朋友通讯系统、同事通讯系统的界面进行设计。功能模块将会为读者讲述如

何设计一个添加功能的模块、删除功能的模块、更新功能的模块、查询功能的模块等。其他模块则包括了帮助文档模块、版本信息模块。

从上面的讲述可以看出整个软件系统是由多个模块构成的,而每个模块可以被设计成一个类,这正是面向对象编程的特色。所谓面向对象编程的方法通俗地说也就是将所有的功能都进行模块化,以方便随时随地调用。这样做的好处就是当某个功能发生变化时,不需要修改整个程序,只须修改功能模块即可。下面将先介绍数据库模块。

20.3.1 数据库模块的设计

1. 建立数据库和数据源

假设读者已掌握 SQL 数据库,所以对于数据库的基础知识将不给予讲述,如果读者不清楚这方面的知识,可以查阅相关的资料。下面将为读者讲述如何设计一个数据库系统。

首先创建一个数据库,然后在数据库中设计几张表,这些表将存储整个软件系统所要显示的数据。这些将会通过 SQL 语句来实现,其实现的步骤如下所示。

01 创建数据库的 SQL 语句如下所示:

```
create database combook
```

02 在 combook 数据库中设计三张表格,分别是 classmate、comfriend、company。创建表格的 SQL 语句如下:

```
Create table classmate  
Create table comfriend  
Create table company
```

03 上面所创建的表格都是空表,下面将为这些表格输入数据,如下所示:

```
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('王鹏','2001',  
'公司','男','上海市浦东','1975-7-20','总经理','30000','3456123-01','1390123456');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('李峰','2002',  
'技术部','男','上海市浦东','1975-2-23','技术总监','10000','3456123-02','137712546');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('王冰','2003',  
'技术部','女','上海徐家汇','1981-2-23','售后部经理','7000','3456123-03','1318123490');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('赵垒','2004',  
'技术部','女','上海南汇区','1980-6-3','工程部经理','7000','3456123-04','1390123452');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('齐衡','2005',  
'技术部','男','上海长宁区','1978-1-18','售后一部主管','5000','3456123-05','1390023465');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('宋丹','2006',  
'技术部','女','上海市浦东','1981-2-23','售后部二主管','5000','3456123-06','1390014456');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('李秀明','2007',  
'商务部','女','上海市浦东','1981-11-9','商务部经理','5000','3456123-07','1390144456');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('赵宾','2008',  
'商务部','女','上海杨浦区','1980-10-27','文员','2000','3456123-08','1390564456');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('陈宾','2009',  
'商务部','男','上海市浦东','1981-2-23','前台','2000','3456123-09','1312312356');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('汤佳','2010',  
'商务部','女','上海普陀区','1982-12-1','商务谈判','3000','3456123-10','1390043216');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('谭华','2011',  
'财务部','女','上海杨浦区','1977-11-2','财务总监','20000','3456123-11','1390563436');  
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('梅亭','2012',  
'财务部','男','上海市浦东','1981-2-23','财务','6000','3456123-12','1312312349');
```



```
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('闫云','2013',
'财务部','女','上海普陀区','1982-12-1','出纳','3000','3456123-13','13900498116');
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('赵飞','2014',
'销售部','女','上海杨浦区','1980-3-27','销售部经理','2000','3456123-14','1390563456');
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('林豪','2015',
'销售部','男','上海市浦东','1981-2-21','销售','2000','3456123-15','1312312346');
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('宋林','2016',
'销售部','女','上海普陀区','1982-12-5','销售','3000','3456123-16','1390043216');
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('王树','2017',
'技术部','女','上海杨浦区','1980-10-27','技术工程师','3000','3456123-17','1390364456');
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('陈豪','2018',
'技术部','男','上海市浦东','1981-2-23','技术工程师','3000','3456123-18','1314123456');
insert into company (name,code,department,sex,address,birthday,duty,salary,tel,motel) values ('汤明','2019',
'技术部','女','上海普陀区','1982-12-1','技术工程师','3000','3456123-19','1390432156');
```

04 上面的 SQL 语句为同事表添加了数据，下面为 comfriend 表输入数据，代码如下所示：

```
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('王昌','男',
'1978-6-10','上海市浦东','上海亮锦科技有限公司','销售部经理','30000','3456123-01','1390123456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('宋玉','女',
'1980-11-20','上海市浦东','上海美文图书工作室','图形策划师','10000','3456123-02','137712546');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('王垒','男',
'1977-7-12','上海徐家汇','上海重横机械厂','书记','7000','3456123-03','1318123490');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('赵庆','男',
'1975-3-12','上海南汇区','上海优化装潢公司','商务','7000','3456123-04','1390123452');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('梅琴','女',
'1985-5-30','上海长宁区','上海斯坦进出口贸易有限公司','报关员','5000','3456123-05','1390023465');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('李丽','女',
'1975-6-10','上海市浦东','大路汽车','销售','5000','3456123-06','1390014456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('陈宸','男',
'1976-8-6','上海市浦东','上海佳键科技有限公司','技术部经理','5000','3456123-07','1390144456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('陈辰','女',
'1976-8-20','上海杨浦区','上海里斯科技有限公司','财务','2000','3456123-08','1390564456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('杨丽','女',
'1980-1-6','上海市浦东','上海荞麦饮食有限公司','厨师长','2000','3456123-09','1312312356');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('杨军','男',
'1975-8-20','上海普陀区','多美发型屋','首席发型师','3000','3456123-10','1390043216');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('郝婷','女',
'1981-6-12','上海杨浦区','利达汽配','前台','20000','3456123-11','1390563436');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('武文','男',
'1978-11-12','上海市浦东','策略游戏公司','游戏设计','6000','3456123-12','1312312349');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('钱燕','女',
'1975-12-20','上海普陀区','大胡子发屋','发型师','3000','3456123-13','13900498116');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('刘意','男',
'1979-7-20','上海杨浦区','中力研究所','研究员','2000','3456123-14','1390563456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('余波','男',
'1965-5-21','上海市浦东','翠微发屋','发型师','2000','3456123-15','1312312346');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('吕菲','女',
'1982-3-11','上海普陀区','上海彰群贸易有限公司','财务','3000','3456123-16','1390043216');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('孙文','男',
'1983-3-3','上海杨浦区','上海莲花生物科技有限公司','技术工程师','3000','3456123-17','1390364456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('周杰','男',
'1984-4-16','上海市浦东','上海未平轮胎公司','副总','3000','3456123-18','1314123456');
insert into comfriend (name,sex,birthday,address,company,duty,salary,tel,phone) values ('政垒','男',
'1981-9-11','上海普陀区','上海梅婷律师事务所','律师','3000','3456123-19','1390432156');
```

05 为 classmate 表格输入数据，代码如下所示：


```

Insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('周胆','男','上海市浦东','上海市浦东','上海',
'上海亮锦科技有限公司','销售部经理','30000','3456123-01','1390123456');
Insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('周冰','女','上海市浦东','上海市浦东','上海',
'上海美文图书工作室','图形策划师','10000','3456123-02','137712546');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('钱郝','男','上海市浦东','上海徐家汇','上海','上海重横机械厂',
'书记','7000','3456123-03','1318123490');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('孙军','男','上海市浦东','上海南汇区','上海','上海优化装潢公司',
'商务','7000','3456123-04','1390123452');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('赵燕','女','上海市浦东','上海长宁区','上海',
'上海斯坦进出口贸易有限公司','报关员','5000','3456123-05','1390023465');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('宋丽丽','女','上海市浦东','上海市浦东','上海','大路汽车',
'销售','5000','3456123-06','1390014456');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('刘军','男','上海市浦东','上海市浦东','上海',
'上海佳键科技有限公司','技术部经理','5000','3456123-07','1390144456');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('陈丽','女','上海市浦东','上海杨浦区','上海',
'上海里斯科技有限公司','财务','2000','3456123-08','1390564456');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('刘菲','女','上海市浦东','上海市浦东','上海',
'上海荠麦饮食有限公司','厨师长','2000','3456123-09','1312312356');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('赵蒙','男','上海市浦东','上海普陀区','上海','多美发型屋',
'首席发型师','3000','3456123-10','1390043216');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('王烨','女','上海市浦东','上海杨浦区','上海','利达汽配',
'前台','20000','3456123-11','1390563436');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('朱意','男','上海市浦东','上海市浦东','上海','策略游戏公司',
'游戏设计','6000','3456123-12','1312312349');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('汤珊','女','上海市浦东','上海普陀区','上海',
'大胡子发屋','发型师','3000','3456123-13','13900498116');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('祝波','男','上海市浦东','上海杨浦区','上海',
'中力研究所','研究员','2000','3456123-14','1390563456');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('段国','男','上海市浦东','上海市浦东','上海',
'翠微发屋','发型师','2000','3456123-15','1312312346');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('王洁','女','上海市浦东','上海普陀区','上海',
'上海彰辟贸易有限公司','财务','3000','3456123-16','1390043216');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('刘烨','男','上海市浦东','上海杨浦区','上海',
'上海莲花生物科技有限公司','技术工程师','3000','3456123-17','1390364456');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('周军','男','上海市浦东','上海市浦东','上海',
'上海未平轮胎公司','副总','3000','3456123-18','1314123456');
insert into classmate (name,sex,address,homeaddress,city,company,duty,salary,contact,homephone)
values ('孙力','男','上海市浦东','上海普陀区','上海',
'上海梅婷律师事务所','律师','3000','3456123-19','1390432156');

```


到此为止，整个数据库的框架已经搭建好，那么通过系统来操作数据库就必须要建立数据源，接下来将为读者介绍如何设定数据源，操作步骤如下：

- 01 选择“开始”|“控制面板”命令，打开如图 20.5 所示界面，双击“管理工具”图标，双击“数据源”选项，如图 20.6 所示。



图 20.5 打开控制面板



图 20.6 双击“数据源”选项

- 02 弹出如图 20.7 所示的界面。由于上例中使用的是 SQL 数据库，而在用户数据源中没有，所以单击“添加”按钮，打开“创建数据源”对话框，如图 20.8 所示。

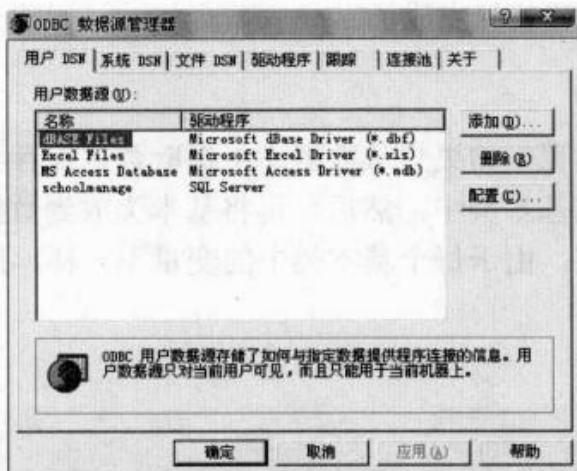


图 20.7 数据源管理器界面



图 20.8 创建新的数据源界面

03 选择 SQL Server 选项，单击“完成”按钮，弹出如图 20.9 所示对话框。

04 填写数据库名称，然后连接数据库服务器，在这里数据库是上面建立好的 combook，由于是本地数据库，所以服务器设置为本地计算机名或者 local。如果是网络数据库，此时必须写入网络服务器名称，或者在下拉列表框中选择服务器。单击“下一步”按钮，出现如图 20.10 所示的对话框。

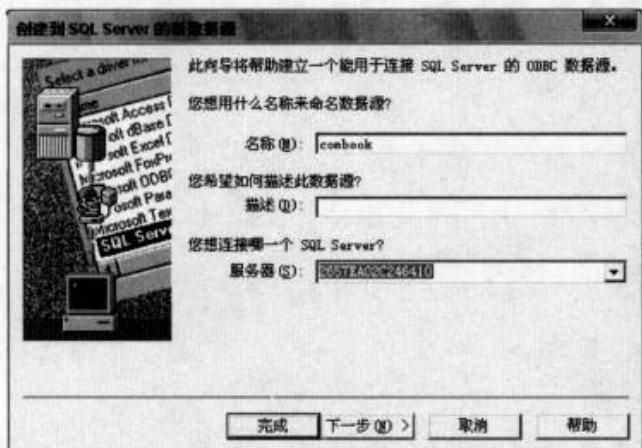


图 20.9 确定数据库名称和服务器名称

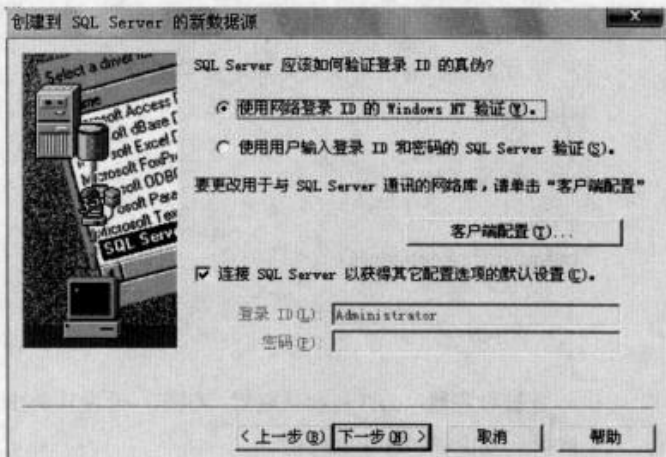


图 20.10 设置登录

05 单击“下一步”按钮，弹出确认对话框，如图 20.11 所示。继续单击“下一步”按钮，最后会出现一个测试界面。如果测试成功，说明创建数据源成功，如图 20.12 所示。

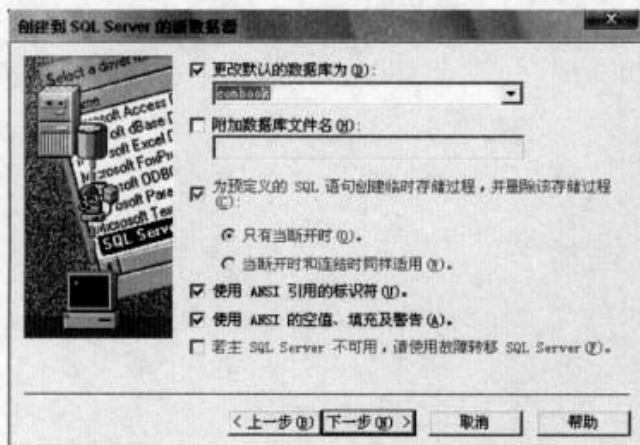


图 20.11 确认数据库名称

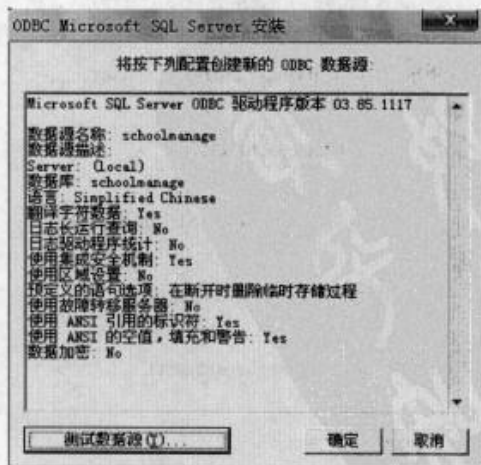


图 20.12 测试数据源

数据源建立完毕，至此整个底层数据库系统的创建就算完成了。

2. 基本类的设计

下面将讲述基本类的设计，为什么要设计基本类呢？如果想从数据库中提取数据保存到 Vector 数据结构中去，首先必须要将这些数据存储到基本类中，然后，再将基本类放到数据结构中去。这些基本类包括同学类、同事类和朋友类。由于每个基本类中的变量不一样，所以下面将列出几个类设计的具体程序代码。

首先将列出同学类设计的基本代码，如下所示：

```
// 同学类的设计
public class classmate
{
    String name;           // 同学的姓名
    String sex;            // 同学的性别
    String homephone;      // 同学的家庭电话
    String address;        // 同学的现在地址
    String company;        // 同学所在的公司
    String duty;           // 同学在公司里的职务
    String salary;         // 同学的薪水
    String contact;        // 同学的个人联系方式
    String homeaddress;    // 同学老家地址
    String city;
    // 通过带参数的构造器来将姓名变量赋值
    classmate(String name)
    {
        this.name=name;
    }
    public String getname()
    {
        return name;
    }
    // 通过设置器（SET）和获取器（GET）来为其余几个变量赋值
    public void setcity(String city)
    {
        this.city=city;
    }
    public void sethomeaddress(String homeaddress)
    {
        this.homeaddress=homeaddress;
    }
    public void setcontact(String contact)
    {
        this.contact=contact;
    }
    public void setinteresting(String interesting)
    {
        this.interesting=interesting;
    }
    public void setsexy(String sex)
    {
        this.sex=sex;
    }
    public void sethomephone(String homephone)
    {

```



```
        this.homephone=homephone;
    }
    public void setaddress(String address)
    {
        this.address=address;
    }
    public void setcompany(String company)
    {
        this.company=company;
    }
    public void setduty(String duty)
    {
        this.duty=duty;
    }
    public void setsalary(String salary)
    {
        this.salary=salary;
    }
    public String getsex()
    {
        return sex;
    }
    public String gethomeaddress()
    {
        return homeaddress;
    }
    public String getaddress()
    {
        return address;
    }
    public String getcompany()
    {
        return company;
    }
    public String getduty()
    {
        return duty;
    }
    public String getsalary()
    {
        return salary;
    }
    public String getcontact()
    {
        return contact;
    }
    public String gethomephone()
    {
        return homephone;
    }
    public String getcity()
    {
        return city;
    }
}
```

```
// 通过 toString()方法可以将类对象以字符串形式输出,在本系统中此方法可以不需要
// 列在这里只是让读者能够熟悉它的用法
```



```

public String toString()
{
    String information="同学姓名: "+name+"性别: "+sex+"所在地址: "+address+"家庭地址: "
        "+homeaddress+"所在城市: "+city+"所在公司: "+company+"职责: "
        "+duty+"薪水: "+salary+"联系方式: "+contact+"家庭电话: "+homephone+"爱好兴趣: "+interesting;
    return information;
}
}

```

下面列出同事类的设计，示例代码如下：

```

public class company
{
    String name;           // 同事的姓名
    String code;           // 同事的工号
    String birthday;       // 同事的生日
    String sex;            // 同事的性别
    String address;        // 同事的家庭地址
    String duty;           // 同事的职务
    String salary;         // 同事的薪水
    String tel;            // 同事的联系电话
    String motel;          // 同事的移动电话
    String department;     // 同事的所在部门
    // 通过构造器来给姓名变量赋值
    company(String name)
    {
        this.name=name;
    }
    public String getname()
    {
        return name;
    }
    public void setcode(String code)
    {
        this.code=code;
    }
    public void setsex(String sex)
    {
        this.sex=sex;
    }
    public void setdepartment(String department)
    {
        this.department=department;
    }
    public void setbirthday(String birthday)
    {
        this.birthday=birthday;
    }
    public void setaddress(String address)
    {
        this.address=address;
    }
    public void setduty(String duty)
    {
        this.duty=duty;
    }
}

```



```
public void setsalary(String salary)
{
    this.salary=salary;
}
public void settel(String tel)
{
    this.tel=tel;
}
public void setmotel(String motel)
{
    this.motel=motel;
}
public String getcode()
{
    return code;
}
public String getsex()
{
    return sex;
}
public String getaddress()
{
    return address;
}
public String getbirthday()
{
    return birthday;
}
public String getdepartment()
{
    return department;
}
public String address()
{
    return address;
}
public String getduty()
{
    return duty;
}
public String getsalary()
{
    return salary;
}
public String gettel()
{
    return tel;
}
public String getmotel()
{
    return motel;
}
public String toString()
{
    String information="同事姓名: "+name+"工号: "+code+"所在部门: "+department+"性别: "
        +sex+"所在地址: "+address+"出生年月: "+birthday+"职责: "+duty+"薪水: "+
```



```
        salary+"联系方式: "+tel+"移动电话: "+motel;  
        return information;  
    }  
}
```

最后介绍朋友类的设计，示例代码如下：

```
public class friend  
{  
    String name;           // 朋友姓名  
    String sex;            // 朋友的性别  
    String birthday;       // 朋友的出生日期  
    String address;        // 朋友的居住地址  
    String company;        // 朋友所在公司  
    String duty;           / 朋友在公司所任职务  
    String salary;         // 朋友的薪水  
    String tel;            // 朋友的电话号码  
    String phone;          // 朋友的手机号码  
    // 通过构造器来给姓名变量赋值  
    friend(String name)  
    {  
        this.name=name;  
    }  
    public String getname()  
    {  
        return name;  
    }  
    public void setsex(String sex)  
    {  
        this.sex=sex;  
    }  
    public void setbirthday(String birthday)  
    {  
        this.birthday=birthday;  
    }  
    public void setaddress(String address)  
    {  
        this.address=address;  
    }  
    public void setcompany(String company)  
    {  
        this.company=company;  
    }  
    public void setduty(String duty)  
    {  
        this.duty=duty;  
    }  
    public void setsalary(String salary)  
    {  
        this.salary=salary;  
    }  
    public void settel(String tel)  
    {  
        this.tel=tel;  
    }  
    public void setphone(String phone)
```



```

{
    this.phone=phone;
}
public String getsex()
{
    return sex;
}
public String getbirthday()
{
    return birthday;
}
public String getaddress()
{
    return address;
}
public String getcompany()
{
    return company;
}
public String getduty()
{
    return duty;
}
public String getsalary()
{
    return salary;
}
public String gettel()
{
    return tel;
}
public String getphone()
{
    return phone;
}
public String toString()
{
    String information="朋友姓名: "+name+"性别: "+sex+"家庭地址: "+address+"出生年月: "+birthday+
        "所在公司: "+company+"职责: "+duty+"薪水: "+salary+"联系方式: "+tel+"家庭电话: "+phone;
    return information;
}
}

```

到此基本类已经建立完毕，下面将为读者介绍如何将数据库中的数据保存到基本类中，然后再将基本类存储到相应的数据结构中。

3. 将数据存储在 Vector 数据结构中

下面主要讲述如何将数据库中的数据提取出来，再存储到 Vector 数据结构中。因为在软件系统中，每个界面都会使用到数据库中的数据，所以将提取数据库中数据的功能也设计成一个类，这样的话，处处都可以调用。

下面首先讲述如何设计同学存储类，其代码如下所示：

```

import java.sql.Connection;
import java.sql.DriverManager;

```



```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;
public class classmatestore
{
    public Vector getclassmate(Connection con,String sql)
    { // 下面的方法将建立一个 Vector 数据结构, 然后提取数据库中的数据存储到 Vector 中
        Vector v=new Vector();
        try
        {
            Statement st=con.createStatement(); // 建立数据库的连接
            ResultSet rs=st.executeQuery(sql);
            while(rs.next())
            {
                // 通过结果集中的 getString 方法从数据库的表中提取表字段的数据
                // 再将提取出来的数据赋值给同学对象。
                // 最后将同学对象存储到 Vector 数据结构中
                String name=rs.getString(1);
                classmate cl=new classmate(name);
                String sexy=rs.getString(2);
                String address=rs.getString(3);
                String homeaddress=rs.getString(4);
                String city=rs.getString(5);
                String company=rs.getString(6);
                String duty=rs.getString(7);
                String salary=rs.getString(8);
                String contact=rs.getString(9);
                String homephone=rs.getString(10);
                cl.setsexy(sexy);
                cl.setaddress(address);
                cl.sethomeaddress(homeaddress);
                cl.setcity(city);
                cl.setcompany(company);
                cl.setduty(duty);
                cl.setsalary(salary);
                cl.setcontact(contact);
                cl.sethomephone(homephone);
                v.add(cl);
            }
            rs.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return v;
    }
    public classmate getobject(Connection con,String name)
    {
        classmate c=null;
        try
        {
            Statement st=con.createStatement();
            String sql="select * from classmate where name='"+name+"'";
            ResultSet rs=st.executeQuery(sql);
```



```
// 通过结果集中的 getString 方法从数据库的表中提取表字段的数据
// 再将提取出来的数据赋值给同学对象
// 最后将同学对象存储到 Vector 数据结构中
while(rs.next())
{
    name=rs.getString(1);
    String sexy=rs.getString(2);
    String address=rs.getString(3);
    String homeaddress=rs.getString(4);
    String city=rs.getString(5);
    String company=rs.getString(6);
    String duty=rs.getString(7);
    String salary=rs.getString(8);
    String contact=rs.getString(9);
    String homephone=rs.getString(10);
    c=new classmate(name);
    c.setsexy(sexy);
    c.setaddress(address);
    c.sethomeaddress(homeaddress);
    c.setcity(city);
    c.setcompany(company);
    c.setduty(duty);
    c.setsalary(salary);
    c.setcontact(contact);
    c.sethomephone(homephone);
}
rs.close();
}
catch(Exception e)
{
    e.printStackTrace();
}
return c;
}

public Connection getConnection()
{ // 通过设置数据库的 URL 密码、用户名来建立与数据库的连接
    Connection con=null;
    String url="jdbc:odbc:combook";
    String username="";
    String password="";
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con=DriverManager.getConnection(url,username,password);
    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    catch(ClassNotFoundException ex)
    {
        ex.printStackTrace();
    }
    return con;
}
```


接下来将讲解如何设计同事存储类，其程序代码如下所示：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;
public class companystore
{
    public Vector getcompany(Connection con,String sql)
    {
        Vector v=new Vector();
        try
        {
            Statement st=con.createStatement();
            ResultSet rs=st.executeQuery(sql);
            while(rs.next())
            {
                // 通过结果集中的 getString 方法从数据库的表中提取表字段的数据
                // 再将提取出来的数据赋值给同事对象
                // 最后将同事对象存储到 Vector 数据结构中
                String name=rs.getString(1);
                company cl=new company(name);
                String code=rs.getString(2);
                String department=rs.getString(3);
                String sex=rs.getString(4);
                String address=rs.getString(5);
                String birthday=rs.getString(6);
                String duty=rs.getString(7);
                String salary=rs.getString(8);
                String tel=rs.getString(9);
                String motel=rs.getString(10);
                cl.setCode(code);
                cl.setDepartment(department);
                cl.setAddress(address);
                cl.setSex(sex);
                cl.setBirthday(birthday);
                cl.setDuty(duty);
                cl.setSalary(salary);
                cl.setTel(tel);
                cl.setMotel(motel);
                v.add(cl);
            }
            rs.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return v;
    }
    public company getObject(Connection con,String name)
    {
        company c=null;
        try
```



```

{
    Statement st=con.createStatement();
    String sql="select * from company where name='"+name+"'";
    ResultSet rs=st.executeQuery(sql);
    // 通过结果集中的 getString 方法从数据库的表中提取表字段的数据
    // 再将提取出来的数据赋值给同事对象。
    // 最后将同事对象存储到 Vector 数据结构中
    while(rs.next())
    {
        name=rs.getString(1);
        String code=rs.getString(2);
        String department=rs.getString(3);
        String sex=rs.getString(4);
        String address=rs.getString(5);
        String birthday=rs.getString(6);
        String duty=rs.getString(7);
        String salary=rs.getString(8);
        String tel=rs.getString(9);
        String motel=rs.getString(10);
        c=new company(name);
        c.setCode(code);
        c.setAddress(address);
        c.setDepartment(department);
        c.setSex(sex);
        c.setBirthday(birthday);
        c.setDuty(duty);
        c.setSalary(salary);
        c.setTel(tel);
        c.setMotel(motel);
    }
    rs.close();
}
catch(Exception e)
{
    e.printStackTrace();
}
return c;
}

public Connection getConnection()
{
    // 通过设置数据库的 URL、密码、用户名来建立与数据库的连接
    Connection con=null;
    String url="jdbc:odbc:combook";
    String username="";
    String password="";
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con=DriverManager.getConnection(url,username,password);
    }
    catch(SQLException e){e.printStackTrace();}
    catch(ClassNotFoundException ex){ex.printStackTrace();}
}
return con;
}
}

```


最后讲述朋友存储类，其程序代码如下所示：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;
public class friendstore
{
    public Vector getfriend(Connection con,String sql)
    {
        Vector v=new Vector();
        try
        {
            Statement st=con.createStatement();
            ResultSet rs=st.executeQuery(sql);
            while(rs.next())
            {
                // 通过结果集中的 getString 方法从数据库的表中提取表字段的数据
                // 再将提取出来的数据赋值给朋友对象
                // 最后将朋友对象存储到 Vector 数据结构中
                String name=rs.getString(1);
                String sexy=rs.getString(2);
                String birthday=rs.getString(3);
                String address=rs.getString(4);
                String company=rs.getString(5);
                String duty=rs.getString(6);
                String salary=rs.getString(7);
                String tel=rs.getString(8);
                String phone=rs.getString(9);
                friend ss=new friend(name);
                ss.setsex(sexy);
                ss.setbirthday(birthday);
                ss.setaddress(address);
                ss.setcompany(company);
                ss.setduty(duty);
                ss.setsalary(salary);
                ss.settel(tel);
                ss.setphone(phone);
                v.add(ss);
            }
            rs.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return v;
    }
    public friend getobject(Connection con,String name)
    {
        friend f=null;
        try
        {
            Statement st=con.createStatement();
```



```

String sql="select * from comfriend where name='"+name+"'";
ResultSet rs=st.executeQuery(sql);
// 通过结果集中的 getString 方法从数据库的表中提取表字段的数据
// 再将提取出来的数据赋值给朋友对象
// 最后将朋友对象存储到 Vector 数据结构中
while(rs.next())
{
    name=rs.getString(1);
    String sexy=rs.getString(2);
    String birthday=rs.getString(3);
    String address=rs.getString(4);
    String company=rs.getString(5);
    String duty=rs.getString(6);
    String salary=rs.getString(7);
    String tel=rs.getString(8);
    String phone=rs.getString(9);
    f=new friend(name);
    f.setsex(sexy);
    f.setbirthday(birthday);
    f.setaddress(address);
    f.setcompany(company);
    f.setduty(duty);
    f.setsalary(salary);
    f.settel(tel);
    f.setphone(phone);
}
rs.close();
}
catch(Exception e)
{
    e.printStackTrace();
}
return f;
}

public Connection getConnection()
{
    // 通过设置数据库的 URL、密码、用户名来建立与数据库的连接
    Connection con=null;
    String url="jdbc:odbc:combook";
    String username="";
    String password="";
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con=DriverManager.getConnection(url,username,password);
    }
    catch(SQLException e){e.printStackTrace();}
    catch(ClassNotFoundException ex){ex.printStackTrace();}
    return con;
}
}

```

以上三个存储类将数据库表中的所有数据存储到了相应的 Vector 数据结构中，以方便操作。到此为止，所有有关数据库方面的设计已经完成。在后面的所有设计工作中遇到有关数据显示的操作，只须直接调用上面三个类即可。

20.3.2 信息界面模块的设计

上面为读者讲述了如何设计数据库，接下来将针对所有的信息界面给予详细讲解，如基本信息系统设计、联系方式系统设计、容器系统设计。其中，每个系统将会针对朋友、同学和同事分别给予详细分析，并且配以程序代码。本节的主要目的是为读者介绍如何使用网格组布局管理器设计复杂的界面，以及如何通过内部类设计事件监听器。

1. 基本信息系统设计

本节将会详细介绍朋友基本信息模块、同事基本信息模块和同学基本信息模块的设计。由于这三个模块的基本框架是一样的，所以放到同一节中讲述。

首先设计一个类，这个类说明了如何设计同学基本信息的界面以及如何将数据结构中的数据提取出来显示到界面上，当然，这里会调用到前面讲过的同学存储类。其详细的程序代码如下所示：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.Connection;
import java.util.Vector;
public class classinfo extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    // 将所有的文本框组件都设定为静态变量，目的是在其他的类中可以自由调用
    // 这里的 SQL 语句的主要目的是将数据库中表的数据提取出来
    // 姓名使用了下拉列表框对象
    String sql="select * from classmate";
    static JComboBox nameinput;
    static JTextField addressinput;
    static JTextField homeaddressinput;
    static JTextField sexyinput;
    static JTextField cityinput;
    static JTextField companyinput;
    static JTextField dutyinput;
    static JTextField salaryinput;
    classmatestore store=new classmatestore();// 下面两句代码的含义是建立数据库的连接
    Connection con=store.getConnection();
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    classinfo()
    {
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
    }
}
```



```

JLabel name=new JLabel("姓名");
JLabel sex=new JLabel("性别");
JLabel address=new JLabel("居住地址");
JLabel homeaddress=new JLabel("家庭地址");
JLabel city=new JLabel("所在城市");
JLabel company=new JLabel("所在公司");
JLabel duty=new JLabel("职位");
JLabel salary=new JLabel("薪水");
nameinput=new JComboBox();
sexinput=new JTextField(10);
addressinput=new JTextField(10);
homeaddressinput=new JTextField(10);
cityinput=new JTextField(10);
companyinput=new JTextField(10);
dutyinput=new JTextField(10);
salaryinput=new JTextField(10);
JLabel title=new JLabel("同学基本信息");
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=4;
constraints.weighty=6;
add(title,constraints,0,0,4,1);
add(name,constraints,0,1,1,1);
add(sex,constraints,0,2,1,1);
add(address,constraints,0,3,1,1);
add(homeaddress,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(sexyinput,constraints,1,2,1,1);
add(addressinput,constraints,1,3,1,1);
add(homeaddressinput,constraints,1,4,1,1);
add(city,constraints,2,1,1,1);
add(company,constraints,2,2,1,1);
add(duty,constraints,2,3,1,1);
add(salary,constraints,2,4,1,1);
add(cityinput,constraints,3,1,1,1);
add(companyinput,constraints,3,2,1,1);
add(dutyinput,constraints,3,3,1,1);
add(salaryinput,constraints,3,4,1,1);
Vector vec=store.getclassmate(con,sql);
// 建立 Vector 数据结构对象, 将存储在其中的基本类对象中数据提取出来
// 再将数据放到文本框组件中去, 在这里通过直接调用前面的同学存储类中的方法而提取数据结构的数据
for(int i=0;i<vec.size();i++)
{
    classmate one =(classmate)vec.get(i);
    String nameselect=one.getname();
    nameinput.addItem(nameselect);
}
String namestring=(String)nameinput.getSelectedItem();
classmate p=store.getobject(con,namestring);
String inputsex=p.getsex();
String inputaddress=p.getaddress();
String inputhomeaddress=p.gethomeaddress();
String inputcity=p.getcity();
String inputcompany=p.getcompany();
String inputduty=p.getduty();
String inputsalary=p.getsalary();

```



```

sexyinput.setText(inputsex);
addressinput.setText(inputaddress);
homeaddressinput.setText(inputhomeaddress);
cityinput.setText(inputcity);
companyinput.setText(inputcompany);
dutyinput.setText(inputduty);
salaryinput.setText(inputsalary);
nameinput.addItemListener(new ItemListener ()
{
    // 当选择下拉列表框中任意项时发生的事件处理, 当选择了任何一个姓名
    // 会将数据结构中的所有数据提取出来, 按照不同的文本框将其放入其中
    public void itemStateChanged(ItemEvent e)
    {
        String namestring=(String)nameinput.getSelectedItem();
        classmate p=store.getobject(con,namestring);
        String inputsex=p.getsex();
        String inputaddress=p.getaddress();
        String inputhomeaddress=p.gethomeaddress();
        String inputcity=p.getcity();
        String inputcompany=p.getcompany();
        String inputduty=p.getduty();
        String inputsalary=p.getsalary();
        sexyinput.setText(inputsex);
        addressinput.setText(inputaddress);
        homeaddressinput.setText(inputhomeaddress);
        cityinput.setText(inputcity);
        companyinput.setText(inputcompany);
        dutyinput.setText(inputduty);
        salaryinput.setText(inputsalary);
    }
});
}
}

```

以上程序代码的运行结果如图 20.13 所示。

同学基本信息	
姓名	刘建
性别	女
所在城市	上海
所在公司	中力研究所
居住地址	上海市浦东
职位	研究员
家庭地址	上海杨浦区
薪水	2000

图 20.13 同学基本信息模块

当打开同学基本信息框架时, 其数据会直接显示出来。因为当运行这个类时, 在这个类的构造器中将存储在 Vector 数据结构中的数据提取出来, 并且显示在界面中的文本框中。

接下来将为读者介绍同事基本信息模块的设计, 其详细的程序代码如下所示:

```

import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.Connection;

```



```
import java.util.Vector;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class companyinfo extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    // 将所有的文本框组件都设定为静态变量，目的是在其他的类中可以自由调用
    // 这里的 SQL 语句的主要目的是将数据库中表的数据提取出来
    // 姓名使用了下拉列表框对象
    String sql="select * from company ";
    static JComboBox nameinput;
    // 下面两句代码的含义是建立数据库的连接
    companystore store=new companystore();
    Connection con=store.getConnection();
    static JTextField codeinput;
    static JTextField sexinput;
    static JTextField departmentinput;
    static JTextField addressinput;
    static JTextField birthdayinput;
    static JTextField dutyinput;
    static JTextField salaryinput;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    companyinfo()
    {
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
        JLabel name=new JLabel("姓名");
        JLabel code=new JLabel("工号");
        JLabel department=new JLabel("部门");
        JLabel sex=new JLabel("性别");
        JLabel address=new JLabel("家庭地址");
        JLabel birthday=new JLabel("出生年月");
        JLabel duty=new JLabel("职位");
        JLabel salary=new JLabel("薪水");
        codeinput=new JTextField(10);
        nameinput=new JComboBox();
        departmentinput=new JTextField(10);
        sexinput=new JTextField(10);
        addressinput=new JTextField(10);
        birthdayinput=new JTextField(10);
        dutyinput=new JTextField(10);
        salaryinput=new JTextField(10);
        JLabel title=new JLabel("同事基本信息");
        GridBagConstraints constraints=new GridBagConstraints();
```



```

constraints.fill=GridBagConstraints.NONE;
constraints.weightx=4;
constraints.weighty=6;
add(title,constraints,0,0,4,1);
add(name,constraints,0,1,1,1);
add(code,constraints,0,2,1,1);
add(department,constraints,0,3,1,1);
add(sex,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(codeinput,constraints,1,2,1,1);
add(departmentinput,constraints,1,3,1,1);
add(sexinput,constraints,1,4,1,1);
add(address,constraints,2,1,1,1);
add(birthday,constraints,2,2,1,1);
add(duty,constraints,2,3,1,1);
    add(salary,constraints,2,4,1,1);
add(addressinput,constraints,3,1,1,1);
add(birthdayinput,constraints,3,2,1,1);
add(dutyinput,constraints,3,3,1,1);
add(salaryinput,constraints,3,4,1,1);
Vector vec=store.getcompany(con,sql);
// 建立 Vector 数据结构对象,将存储在其中的基本类对象中的数据提取出来
// 再将数据放到文本框组件中去,在这里通过直接调用前面的同事存储类中的方法而提取数据结构的数据
for(int i=0;i<vec.size();i++)
{
    company one =(company)vec.get(i);
    String nameselect=one.getname();
    nameinput.addItem(nameselect);
}
String namestring=(String)nameinput.getSelectedItem();
company p=store.getobject(con,namestring);
String inputcode=p.getcode();
String inputsex=p.getsex();
String inputbirthday=p.getbirthday();
String inputdepartment=p.getdepartment();
String inputaddress=p.getaddress();
String inputduty=p.getduty();
String inputsalary=p.getsalary();
sexinput.setText(inputsex);
codeinput.setText(inputcode);
birthdayinput.setText(inputbirthday);
addressinput.setText(inputaddress);
departmentinput.setText(inputdepartment);
dutyinput.setText(inputduty);
salaryinput.setText(inputsalary);
nameinput.addItemListener(new ItemListener ()
{
    // 当选择下拉列表框中任意项时发生的事件处理,当选择任何一个姓名
    // 会将数据结构中的所有数据提取出来,按照不同的文本框将其放入其中
    public void itemStateChanged(ItemEvent e)
    {
        String namestring=(String)nameinput.getSelectedItem();
        company p=store.getobject(con,namestring);
        String inputcode=p.getcode();
        String inputsex=p.getsex();
        String inputbirthday=p.getbirthday();
        String inputdepartment=p.getdepartment();
    }
}

```



```

String inputaddress=p.getaddress();
String inputduty=p.getduty();
String inputsalary=p.getsalary();
sexinput.setText(inputsex);
codeinput.setText(inputcode);
birthdayinput.setText(inputbirthday);
addressinput.setText(inputaddress);
departmentinput.setText(inputdepartment);
dutyinput.setText(inputduty);
salaryinput.setText(inputsalary);
    }
    });
}
}

```

以上程序代码的运行结果如图 20.14 所示。

同事基本信息			
姓名	王鹏	家庭地址	上海市浦东
工号	20011	出生年月	1975-7-20
部门	公司	职位	总经理 1
性别	男	薪水	30000

图 20.14 同事基本信息模块

同样，当打开同事基本信息框架时，其数据会直接显示出来。最后将为读者介绍朋友基本信息模块的设计，其详细程序代码如下所示：

```

import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.Connection;
import java.util.Vector;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class friendinfo extends JPanel
{
    private static final long serialVersionUID = 1L;
    static JComboBox nameinput;
    friendstore store=new friendstore();
    Connection con=store.getConnection();
    static JTextField sexyinput;
    static JTextField birthdayinput;
    static JTextField addressinput;
    static JTextField companyinput;
    static JTextField dutyinput;
    static JTextField salaryinput;
    String sql="select * from comfriend";
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {

```



```

constraints.gridx=x;
constraints.gridy=y;
constraints.gridwidth=w;
constraints.gridheight=h;
add(c,constraints);
}
friendinfo()
{
    GridBagLayout lay=new GridBagLayout();
    setLayout(lay);
    JLabel name=new JLabel("姓名");
    JLabel sex=new JLabel("性别");
    JLabel birthday=new JLabel("出生年月");
    JLabel address=new JLabel("家庭地址");
    JLabel company=new JLabel("所在公司");
    JLabel duty=new JLabel("职位");
    JLabel salary=new JLabel("薪水");
    salaryinput=new JTextField(10);
    nameinput=new JComboBox();
    sexyinput=new JTextField(10);
    birthdayinput=new JTextField(10);
    addressinput=new JTextField(10);
    companyinput=new JTextField(20);
    dutyinput=new JTextField(10);
    JLabel title=new JLabel("朋友基本信息");
    GridBagConstraints constraints=new GridBagConstraints();
    constraints.fill=GridBagConstraints.NONE;
    constraints.weightx=4;
    constraints.weighty=6;
    add(title,constraints,0,0,4,1);
    add(name,constraints,0,1,1,1);
    add(sex,constraints,0,2,1,1);
    add(birthday,constraints,0,3,1,1);
    add(address,constraints,0,4,1,1);
    add(nameinput,constraints,1,1,1,1);
    add(sexyinput,constraints,1,2,1,1);
    add(birthdayinput,constraints,1,3,1,1);
    add(addressinput,constraints,1,4,1,1);
    add(company,constraints,2,1,1,1);
    add(duty,constraints,2,2,1,1);
    add(salary,constraints,2,3,1,1);
    add(companyinput,constraints,3,1,1,1);
    add(dutyinput,constraints,3,2,1,1);
    add(salaryinput,constraints,3,3,1,1);
    Vector vec=store.getfriend(con,sql);
    for(int i=0;i<vec.size();i++)
    // 建立 Vector 数据结构对象, 将存储在其中的基本类对象中的数据提取出来
    // 再将数据放到文本框组件中去, 在这里通过直接调用前面的朋友存储类中的方法而提取数据结构的数据
    {
        friend one =(friend)vec.get(i);
        String nameselect=one.getname();
        nameinput.addItem(nameselect);
    }
    String namestring=(String)nameinput.getSelectedItem();
    friend p=store.getobject(con,namestring);
    String inputsex=p.getsex();

```



```
String inputbirthday=p.getbirthday();
String inputaddress=p.getaddress();
String inputcompany=p.getcompany();
String inputduty=p.getduty();
String inputsalary=p.getsalary();
sexyinput.setText(inputsex);
birthdayinput.setText(inputbirthday);
addressinput.setText(inputaddress);
companyinput.setText(inputcompany);
dutyinput.setText(inputduty);
salaryinput.setText(inputsalary);
nameinput.addItemListener(new ItemListener ()
{ // 当选择下拉列表框中任意项时发生的事件处理, 当选择任何一个姓名
  // 会将数据结构中的所有数据提取出来, 按照不同的文本框将其放入其中
  public void itemStateChanged(ItemEvent e)
  {
    String namestring=(String)nameinput.getSelectedItem();
    friend p=store.getobject(con,namestring);
    String inputsex=p.getsex();
    String inputbirthday=p.getbirthday();
    String inputaddress=p.getaddress();
    String inputcompany=p.getcompany();
    String inputduty=p.getduty();
    String inputsalary=p.getsalary();
    sexyinput.setText(inputsex);
    birthdayinput.setText(inputbirthday);
    addressinput.setText(inputaddress);
    companyinput.setText(inputcompany);
    dutyinput.setText(inputduty);
    salaryinput.setText(inputsalary);
  }
});
}
```

上面程序的运行结果如图 20.15 所示。

朋友基本信息			
姓名	王磊	所在公司	上海亮棉科技有限公司
性别	男11	职位	销售部经理
出生年月	1978-6-10	薪水	30000 1
家庭地址	上海市浦东		

图 20.15 朋友基本信息模块

仍然与前面两个模块一样, 打开框架数据就会显示出来。至此, 基本信息模块就讲述完毕了。既然是通讯录系统, 还需要有联系方式, 下面将为读者进行介绍。

2. 联系方式模块设计

下面将介绍联系方式模块的设计, 其中包括同学联系方式模块、同事联系方式模块和朋友联系方式模块。下面首先介绍同学联系方式模块的设计, 其程序代码如下所示。


```
import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.Connection;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class classmatecommunication
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static JComboBox nameinput;
    String sql="select * from classmate ";
    classmatestore store=new classmatestore();
    Connection con=store.getConnection();
    static JTextField homephonetext;
    // 将所有的文本框组件都设定为静态变量，目的是在其他的类中可以自由调用
    static JTextField moteltext;
    static JLabel name;
    static JLabel title;
    static JLabel title1;
    static JButton closebutton;
    static JPanel pane;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        pane.add(c,constraints);
    }
    classmatecommunication()
    {
        pane=new JPanel();
        GridBagLayout lay=new GridBagLayout();
        pane.setLayout(lay);
        GridBagConstraints constraints=new GridBagConstraints();
        constraints.fill=GridBagConstraints.NONE;
        constraints.weightx=5;
        constraints.weighty=3;
        name=new JLabel("姓名");
        nameinput=new JComboBox();
        title=new JLabel("家庭电话号码");
        title1=new JLabel("个人移动电话号码");
```



```

homephonetext=new JTextField(20);
moteltext=new JTextField(20);
closebutton=new JButton("关闭此窗口");
add(title,constraints,1,0,3,1);
add(homephonetext,constraints,1,1,3,1);
add(title1,constraints,1,2,3,1);
add(moteltext,constraints,1,3,3,1);
add(closebutton,constraints,2,4,3,1);
add(title,constraints,1,1,3,1);
add(homephonetext,constraints,1,2,3,1);
add(title1,constraints,1,3,3,1);
add(moteltext,constraints,1,4,3,1);
add(closebutton,constraints,2,5,3,1);
add(name,constraints,1,0,1,1);
add(nameinput,constraints,2,0,1,1);
Vector vec=store.getClassmate(con,sql);
// 将 Vector 数据结构中的数据提取出来, 赋给文本框的文本属性
for(int i=0;i<vec.size();i++)
{
    classmate one =(classmate)vec.get(i);
    String nameselect=one.getname();
    nameinput.addItem(nameselect);
}
String namestring=(String)nameinput.getSelectedItem();
classmate p=store.getObject(con,namestring);
String inputtel=p.getcontact();
String inputmotel=p.gethomephone();
homephonetext.setText(inputtel);
moteltext.setText(inputmotel);
nameinput.addItemListener(new ItemListener ()
{
    // 当单击下拉列表框中的姓名选项, 会在文本框中显示出相应姓名的电话号码和手机号码
    public void itemStateChanged(ItemEvent e)
    {
        String namestring=(String)nameinput.getSelectedItem();
        classmate p=store.getObject(con,namestring);
        String inputtel=p.getcontact();
        String inputmotel=p.gethomephone();
        homephonetext.setText(inputtel);
        moteltext.setText(inputmotel);
    }
});
closebutton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        pane.setVisible(false);
    }
});
pane.setVisible(false);
}
}

```

上面程序代码的运行结果如图 20.16 所示。



图 20.16 同学联系方式模块

与基本信息模块一样，当选择一个姓名后，其通讯方式就会自动显示出来。接下来将介绍同事联系方式模块的设计。其程序代码如下所示：

```
import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.Connection;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class companycommunication
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static JComboBox nameinput;
    String sql="select * from company ";
    companystore store=new companystore();
    Connection con=store.getConnection();
    static JTextField homephonetext;
    static JTextField moteltext;
    static JLabel title;
    static JLabel title1;
    static JLabel name;
    static JButton closebutton;
    static JPanel pane;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        pane.add(c,constraints);
    }
}
companycommunication()
```



```

{
    pane=new JPanel();
    GridBagLayout lay=new GridBagLayout();
    pane.setLayout(lay);
    GridBagConstraints constraints=new GridBagConstraints();
    constraints.fill=GridBagConstraints.NONE;
    constraints.weightx=6;
    constraints.weighty=6;
    title=new JLabel("公司电话号码");
    title1=new JLabel("个人电话号码");
    name=new JLabel("姓名");
    nameinput=new JComboBox();
    homephonetext=new JTextField(20);
    moteltext=new JTextField(20);
    closebutton=new JButton("关闭此窗口");
    add(title,constraints,1,1,3,1);
    add(homephonetext,constraints,1,2,3,1);
    add(title1,constraints,1,3,3,1);
    add(moteltext,constraints,1,4,3,1);
    add(closebutton,constraints,2,5,3,1);
    add(name,constraints,1,0,1,1);
    add(nameinput,constraints,2,0,1,1);
    Vector vec=store.getcompany(con,sql);
    // 从数据结构中的基本类中提取数据, 再将数据放到文本框中
    for(int i=0;i<vec.size();i++)
    {
        company one =(company)vec.get(i);
        String nameselect=one.getname();
        nameinput.addItem(nameselect);
    }
    String namestring=(String)nameinput.getSelectedItem();
    company p=store.getobject(con,namestring);
    String inputtel=p.gettel();
    String inputmotel=p.getmotel();
    homephonetext.setText(inputtel);
    moteltext.setText(inputmotel);
    nameinput.addItemListener(new ItemListener ()
    { // 处理下拉文本框事件, 当单击下拉列表框的姓名选项时,
      // 会在文本框中显示出相应姓名的电话号码和手机号码信息
      public void itemStateChanged(ItemEvent e)
      {
          String namestring=(String)nameinput.getSelectedItem();
          company p=store.getobject(con,namestring);
          String inputtel=p.gettel();
          String inputmotel=p.getmotel();
          homephonetext.setText(inputtel);
          moteltext.setText(inputmotel);
      }
    });
    closebutton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent Event)
        {
            pane.setVisible(false);
        }
    });
}

```



```
pane.setVisible(false);
}
}
```

上面程序代码的运行结果如图 20.17 所示。

图 20.17 同事联系方式模块

最后，将为读者介绍朋友联系方式模块的设计，其程序代码如下所示：

```
import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.Connection;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class friendcommunication
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static JComboBox nameinput;
    String sql="select * from comfriend ";
    friendstore store=new friendstore();
    Connection con=store.getConnection();
    static JTextField homephonetext;
    static JTextField moteltext;
    static JLabel name;
    static JLabel title;
    static JButton closebutton;
    static JLabel title1;
    static JPanel pane;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
```



```

constraints.gridwidth=w;
constraints.gridheight=h;
pane.add(c,constraints);
}
friendcommunication()
{
    pane=new JPanel();
    GridBagLayout lay=new GridBagLayout();
    pane.setLayout(lay);
    GridBagConstraints constraints=new GridBagConstraints();
    constraints.fill=GridBagConstraints.NONE;
    constraints.weightx=6;
    constraints.weighty=6;
    nameinput=new JComboBox();
    name=new JLabel("姓名");
    title=new JLabel("家乡家庭电话号码");
    title1=new JLabel("目前个人电话号码");
    homephonetext=new JTextField(20);
    moteltext=new JTextField(20);
    closebutton=new JButton("关闭此窗口");
    add(title,constraints,1,1,3,1);
    add(homephonetext,constraints,1,2,3,1);
    add(title1,constraints,1,3,3,1);
    add(moteltext,constraints,1,4,3,1);
    add(closebutton,constraints,2,5,3,1);
    add(name,constraints,1,0,1,1);
    add(nameinput,constraints,2,0,1,1);
    Vector vec=store.getfriend(con,sql);
    for(int i=0;i<vec.size();i++)
    {
        friend one =(friend)vec.get(i);
        String nameselect=one.getname();
        nameinput.addItem(nameselect);
    }
    String namestring=(String)nameinput.getSelectedItem();
    friend p=store.getobject(con,namestring);
    String inputtel=p.gettel();
    String inputmotel=p.getphone();
    homephonetext.setText(inputtel);
    moteltext.setText(inputmotel);
    nameinput.addItemListener(new ItemListener ()
    { // 当单击下拉列表框中的姓名选项时，会在文本框中显示出相应姓名的电话号码和手机号码
        public void itemStateChanged(ItemEvent e)
        {
            String namestring=(String)nameinput.getSelectedItem();
            friend p=store.getobject(con,namestring);
            String inputtel=p.gettel();
            String inputmotel=p.getphone();
            homephonetext.setText(inputtel);
            moteltext.setText(inputmotel);
        }
    });
    closebutton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent Event)
        {

```



```
pane.setVisible(false);  
}  
});  
pane.setVisible(false);  
}  
}
```

上面程序代码的运行结果如图 20.18 所示。

图 20.18 朋友联系方式模块

这样就可以通过姓名的选择来查询其通讯方式，其实这个模块也可以作为查询模块来使用，只不过不太方便，通过本小节，读者可以思考一下，如果设计一个学校管理系统，如何通过选择一个学生的学号或者一个老师的工号，显示出此学生或者老师的全部信息。希望读者能够按照以上的思路自行给予试验。

20.3.3 功能模块的设计

一个专业的信息系统都必须具备几个功能：添加、删除、查询和更新。同样，将这些功能设计成一个一个的类，因为这样的设计，会让主程序显得简洁，也使得主程序的可读性强，甚至在以后想增加其他模块，需要添加这些功能时就无须重新编写，只须直接调用即可。接下来将会为读者分别讲述如何设计它们。

1. 添加功能模块的设计

下面将为读者讲述如何设计添加功能模块类，要设计添加功能模块必须要达到两个目的，其一就是让添加的数据录入底层数据库中，其二是让添加的数据立即在界面中体现出来。下面先来观察同学信息系统中的添加功能类，其程序代码如下所示：

```
import java.awt.Component;  
import java.awt.Dimension;  
import java.awt.GridBagConstraints;  
import java.awt.GridBagLayout;  
import java.awt.Toolkit;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.sql.Connection;  
import java.sql.Statement;  
import java.util.Vector;  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JLabel;
```



```

import javax.swing.JPanel;
import javax.swing.JTextField;
public class addclassmate extends JPanel
{
    private static final long serialVersionUID = 1L;
    final JTextField nameinput;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JFrame frame;
    static classmate ss;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    // 此方法的含义是在容器中添加组件，并且按照网格组布局管理器的规则添加
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    addclassmate()
    {
        frame=new JFrame("同学信息添加窗口");
        frame.setContentPane(this);
        Toolkit kit=Toolkit.getDefaultToolkit();// 确定窗口框架的显示位置和大小
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setVisible(true);
        frame.setSize(WIDTH,HEIGHT);
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
        JLabel name=new JLabel("姓名");
        JLabel sex=new JLabel("性别");
        JLabel address=new JLabel("居住地址");
        JLabel homeaddress=new JLabel("家庭地址");
        JLabel city=new JLabel("所在城市");
        JLabel company=new JLabel("所在公司");
        JLabel duty=new JLabel("职位");
        JLabel salary=new JLabel("薪水");
        JLabel contact=new JLabel("移动电话");
        JLabel homephone=new JLabel("家庭电话");
        nameinput=new JTextField(10);
        final JTextField sexyinput=new JTextField(10);
        final JTextField addressinput=new JTextField(10);
        final JTextField homeaddressinput=new JTextField(10);
        final JTextField cityinput=new JTextField(10);
        final JTextField companyinput=new JTextField(10);
        final JTextField dutyinput=new JTextField(10);
        final JTextField salaryinput=new JTextField(10);
        final JTextField contactinput=new JTextField(10);
        final JTextField homephoneinput=new JTextField(10);
        JLabel title=new JLabel("同学基本信息");
        JButton addbutton=new JButton("添加同学信息数据");
    }
}

```



```

GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=4;
constraints.weighty=6;
add(title,constraints,0,0,4,1);
add(name,constraints,0,1,1,1);
add(sex,constraints,0,2,1,1);
add(address,constraints,0,3,1,1);
add(homeaddress,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(sexyinput,constraints,1,2,1,1);
add(addressinput,constraints,1,3,1,1);
add(homeaddressinput,constraints,1,4,1,1);
add(city,constraints,2,1,1,1);
add(company,constraints,2,2,1,1);
add(duty,constraints,2,3,1,1);
add(salary,constraints,2,4,1,1);
add(cityinput,constraints,3,1,1,1);
add(companyinput,constraints,3,2,1,1);
add(dutyinput,constraints,3,3,1,1);
add(salaryinput,constraints,3,4,1,1);
add(contact,constraints,0,5,1,1);
add(homephone,constraints,2,5,1,1);
add(contactinput,constraints,1,5,1,1);
add(homephoneinput,constraints,3,5,1,1);
add(addbutton,constraints,0,6,2,1);
addbutton.addActionListener(new ActionListener()
{ // 单击“添加”按钮后，会将数据保存到 Vector 数据结构中，再从数据结构中将数据保存到数据库中，
  // 同时会将新添加的数据添加到基本信息界面中
  public void actionPerformed(ActionEvent Event)
  {
    try
    {
      String nametext=nameinput.getText();
      String sextext=sexyinput.getText();
      String addresstext=addressinput.getText();
      String homeaddresstext=homeaddressinput.getText();
      String citytext=cityinput.getText();
      String companytext=companyinput.getText();
      String dutytext=dutyinput.getText();
      String salarytext=salaryinput.getText();
      String contacttext=contactinput.getText();
      String homephonetext=homephoneinput.getText();
      classmatestore store=new classmatestore();
      Connection con=store.getConnection();
      Statement st=con.createStatement();
      String sql="insert into classmate values('"+nametext+"','"+sextext+"','"+addresstext+"',
        '"+homeaddresstext+"','"+citytext+"','"+companytext+"','"+dutytext+"','"+salarytext+"',
        '"+contacttext+"','"+homephonetext+"')";
      st.executeUpdate(sql);
      ss=new classmate(nametext);
      ss.setsexy(sextext);
      ss.setaddress(addresstext);
      ss.sethomeaddress(homeaddresstext);
      ss.setcity(citytext);
      ss.setcompany(companytext);
    }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
  }
});

```



```

ss.setduty(dutytext);
ss.setsalary(salarytext);
ss.setcontact(contacttext);
ss.sethomephone(homephonetext);
Vector vec=new Vector();
vec.add(ss);
classinfo.nameinput.addItem(nametext);
classinfo.nameinput.setSelectedItem(nametext);
classinfo.sexyinput.setText(sextext);
classinfo.addressinput.setText(addresstext);
classinfo.homeaddressinput.setText(homeaddresstext);
classinfo.companyinput.setText(companytext);
classinfo.dutyinput.setText(dutytext);
classinfo.salaryinput.setText(salarytext);
}
catch(Exception e){}
frame.dispose();
}
});
}
}

```

上面程序代码的运行结果如图 20.19 所示。

图 20.19 同学添加功能模块

当文本框中输入了数据后，单击“添加同学信息数据”按钮后，窗口会自动关闭，并且会把数据添加到数据库中。接下来为读者介绍同事添加功能模块，其程序代码如下所示：

```

import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.Statement;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class addcompany extends JPanel
{

```



```
private static final long serialVersionUID = 1L;
final JTextField nameinput;
static final int WIDTH=600;
static final int HEIGHT=300;
final JTextField departmentinput;
final JTextField sexinput;
final JTextField addressinput;
final JTextField birthdayinput;
final JTextField dutyinput;
final JTextField salaryinput;
final JTextField telinput;
final JTextField motelinput;
JFrame frame;
public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
{
    constraints.gridx=x;
    constraints.gridy=y;
    constraints.gridwidth=w;
    constraints.gridheight=h;
    add(c,constraints);
}
addcompany()
{
    frame=new JFrame("添加同事信息窗口");
    frame.setContentPane(this);
    Toolkit kit=Toolkit.getDefaultToolkit();
    Dimension screenSize=kit.getScreenSize();
    int width=screenSize.width;
    int height=screenSize.height;
    int x=(width-WIDTH)/2;
    int y=(height-HEIGHT)/2;
    frame.setLocation(x,y);
    frame.setVisible(true);
    frame.setSize(WIDTH,HEIGHT);
    GridBagLayout lay=new GridBagLayout();
    setLayout(lay);
    JLabel name=new JLabel("姓名");
    JLabel code=new JLabel("工号");
    JLabel department=new JLabel("部门");
    JLabel sex=new JLabel("性别");
    JLabel address=new JLabel("家庭地址");
    JLabel birthday=new JLabel("出生年月");
    JLabel duty=new JLabel("职位");
    JLabel salary=new JLabel("薪水");
    JLabel tel=new JLabel("公司电话");
    JLabel motel=new JLabel("移动电话");
    final JTextField codeinput=new JTextField(10);
    nameinput=new JTextField(10);
    departmentinput=new JTextField(10);
    sexinput=new JTextField(10);
    addressinput=new JTextField(10);
    birthdayinput=new JTextField(10);
    dutyinput=new JTextField(10);
    salaryinput=new JTextField(10);
    telinput=new JTextField(10);
    motelinput=new JTextField(10);
```



```

JLabel title=new JLabel("同事基本信息");
JButton addbutton=new JButton("添加同事信息数据");
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=4;
constraints.weighty=6;
add(title,constraints,0,0,4,1);           // 使用网格组布局添加控件
add(name,constraints,0,1,1,1);
add(code,constraints,0,2,1,1);
add(department,constraints,0,3,1,1);
add(sex,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(codeinput,constraints,1,2,1,1);
add(departmentinput,constraints,1,3,1,1);
add(sexinput,constraints,1,4,1,1);
add(address,constraints,2,1,1,1);
add(birthday,constraints,2,2,1,1);
add(duty,constraints,2,3,1,1);
add(salary,constraints,2,4,1,1);
add(addressinput,constraints,3,1,1,1);
add(birthdayinput,constraints,3,2,1,1);
add(dutyinput,constraints,3,3,1,1);
add(salaryinput,constraints,3,4,1,1);
add(tel,constraints,0,5,1,1);
add(motel,constraints,2,5,1,1);
add(telinput,constraints,1,5,1,1);
add(motelinput,constraints,3,5,1,1);
add(addbutton,constraints,0,6,2,1);
addbutton.addActionListener(new ActionListener()
{
    // 单击“添加”按钮后，会将数据保存到 Vector 数据结构中，再从数据结构中将数据保存到数据库中，
    // 同时，会将新添加的数据添加到基本信息界面中。
    public void actionPerformed(ActionEvent Event)
    {
        try
        {
            String nametext=nameinput.getText();
            String codetext=codeinput.getText();
            String departmenttext=departmentinput.getText();
            String sextext=sexinput.getText();
            String birthdaytext=birthdayinput.getText();
            String addresstext=addressinput.getText();
            String salarytext=salaryinput.getText();
            String dutytext=dutyinput.getText();
            String moteltext=motelinput.getText();
            String teltext=telinput.getText();
            companystore store=new companystore();
            Connection con=store.getConnection();
            Statement st=con.createStatement();
            String sql="insert into company values('"+nametext+"',
                '"+codetext+"','"+departmenttext+"','"+sextext+"','"+addresstext+"','"+birthdaytext+"',
                '"+dutytext+"','"+salarytext+"','"+teltext+"','"+moteltext+"')";
            st.executeUpdate(sql);
            company ss=new company(nametext);
            ss.setCode(codetext);
            ss.setDepartment(departmenttext);
            ss.setSex(sextext);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
});

```



```

        ss.setbirthday(birthdaytext);
        ss.setduty(dutytext);
        ss.setsalary(salarytext);
        ss.settel(teltext);
        ss.setmotel(moteltext);
        Vector vec=new Vector();
        vec.add(ss);
        companyinfo.nameinput.addItem(nametext);
        companyinfo.nameinput.setSelectedItem(nametext);
        companyinfo.codeinput.setText(codetext);
        companyinfo.departmentinput.setText(departmenttext);
        companyinfo.sexinput.setText(sextext);
        companyinfo.birthdayinput.setText(birthdaytext);
        companyinfo.dutyinput.setText(dutytext);
        companyinfo.salaryinput.setText(salarytext);
    }
    catch(Exception e){}
    frame.dispose();
}
});
}
}

```

上面程序代码的运行结果如图 20.20 所示。

图 20.20 同事添加功能模块类

同事添加功能模块类与同学添加功能模块类相似。最后，将为读者讲述朋友添加功能模块类的设计，其程序代码如下所示：

```

import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.Statement;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class addfriend extends JPanel

```



```
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    final JTextField sexyinput;
    final JTextField birthdayinput;
    final JTextField addressinput;
    final JTextField companyinput;
    final JTextField dutyinput;
    final JTextField salaryinput;
    final JTextField nameinput;
    JTextField telinput;
    JTextField phoneinput;
    static final int WIDTH=600;
    static final int HEIGHT=300;
    JFrame frame;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    addfriend()
    {
        frame=new JFrame("添加朋友信息窗口");
        frame.setContentPane(this);
        Toolkit kit=Toolkit.getDefaultToolkit();
        Dimension screenSize=kit.getScreenSize();
        int width=screenSize.width;
        int height=screenSize.height;
        int x=(width-WIDTH)/2;
        int y=(height-HEIGHT)/2;
        frame.setLocation(x,y);
        frame.setSize(WIDTH,HEIGHT);
        frame.setVisible(true);
        GridBagLayout lay=new GridBagLayout();
        setLayout(lay);
        JLabel name=new JLabel("姓名");
        JLabel sex=new JLabel("性别");
        JLabel birthday=new JLabel("出生年月");
        JLabel address=new JLabel("家庭地址");
        JLabel company=new JLabel("所在公司");
        JLabel duty=new JLabel("职位");
        JLabel salary=new JLabel("薪水");
        JLabel tel=new JLabel("个人联系方式");
        JLabel phone=new JLabel("家庭电话");
        salaryinput=new JTextField(10);
        nameinput=new JTextField(10);
        sexyinput=new JTextField(10);
        birthdayinput=new JTextField(10);
        addressinput=new JTextField(10);
        companyinput=new JTextField(10);
        dutyinput=new JTextField(10);
    }
}
```



```

telinput=new JTextField(10);
phoneinput=new JTextField(10);
JLabel title=new JLabel("朋友基本信息");
JButton addbutton=new JButton("添加朋友信息数据");
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=4;
constraints.weighty=6;
add(title,constraints,0,0,4,1);           // 使用网格组布局添加控件
add(name,constraints,0,1,1,1);
add(sex,constraints,0,2,1,1);
add(birthday,constraints,0,3,1,1);
add(address,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(sexyinput,constraints,1,2,1,1);
add(birthdayinput,constraints,1,3,1,1);
add(addressinput,constraints,1,4,1,1);
add(company,constraints,2,1,1,1);
add(duty,constraints,2,2,1,1);
add(salary,constraints,2,3,1,1);
add(companyinput,constraints,3,1,1,1);
add(dutyinput,constraints,3,2,1,1);
add(salaryinput,constraints,3,3,1,1);
add(tel,constraints,0,5,1,1);
add(phone,constraints,2,4,1,1);
add(telinput,constraints,1,5,1,1);
add(phoneinput,constraints,3,4,1,1);
add(addbutton,constraints,0,6,2,1);
addbutton.addActionListener(new ActionListener()
{
    // 单击“添加”按钮后，会将数据保存到 Vector 数据结构中，再从数据结构中将数据保存到数据库中，
    // 同时，会将新添加的数据添加到基本信息界面中
    public void actionPerformed(ActionEvent Event)
    {
        try
        {
            String nametext=nameinput.getText();
            String sextext=sexyinput.getText();
            String birthdaytext=birthdayinput.getText();
            String addresstext=addressinput.getText();
            String companytext=companyinput.getText();
            String dutytext=dutyinput.getText();
            String salarytext=salaryinput.getText();
            String teltext=telinput.getText();
            String phonetext=phoneinput.getText();
            friendstore store=new friendstore();
            Connection con=store.getConnection();
            Statement st=con.createStatement();
            String sql="insert into comfriend
                values('"+nametext+"','"+sextext+"','"+birthdaytext+"','"+addresstext+"','"+
                companytext+"','"+dutytext+"','"+salarytext+"','"+teltext+"','"+phonetext+"')";
            st.executeUpdate(sql);
            friend ss=new friend(nametext);
            ss.setsex(sextext);
            ss.setbirthday(birthdaytext);
            ss.setaddress(addresstext);
            ss.setcompany(companytext);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
});

```



```

        ss.setduty(dutytext);
        ss.setsalary(salarytext);
        ss.settel(teltext);
        ss.setphone(phonetext);
        Vector vec=new Vector();
        vec.add(ss);
        friendinfo.nameinput.addItem(nametext);
        friendinfo.nameinput.setSelectedItem(nametext);
        friendinfo.sexyinput.setText(sextext);
        friendinfo.birthdayinput.setText(birthdaytext);
        friendinfo.addressinput.setText(addresstext);
        friendinfo.companyinput.setText(companytext);
        friendinfo.dutyinput.setText(dutytext);
        friendinfo.salaryinput.setText(salarytext);
    }
    catch(Exception e){}
    frame.dispose();
}
});
}
public static void main(String[] args)
{
    new addfriend();
}
}

```

上面程序代码的运行结果如图 20.21 所示。

图 20.21 朋友添加功能模块类的设计

到此为止，添加功能就讲述完了，下面将讲述删除功能的设计。

2. 删除功能模块类的设计

删除功能主要是要达到两个目的：其一是要从数据库中删除数据；其二就是要从界面中删除数据。首先来看看同学删除功能类的设计，其程序代码如下所示：

```

import java.sql.Connection;
import java.sql.Statement;
public class delclassmate
{
    delclassmate()
    {
        String namestring=(String)classinfo.nameinput.getSelectedItemAt();
        classmatestore store=new classmatestore();
        try

```



```
{
    Connection con=store.getConnection();
    Statement st=con.createStatement();
    String sql="delete from classmate where name='"+namestring+"'";
    st.executeUpdate(sql);
}
catch(Exception e){}
classinfo.nameinput.removeItem(namestring);
}
```

同样，同事的删除功能类和朋友的删除功能类与此类基本相同，惟一不同的是因为它们所述的父窗口不同和数据库中表不同。下面再给出同事删除功能类的程序代码：

```
import java.sql.Connection;
import java.sql.Statement;
public class delcompany
{
    delcompany()
    {
        String namestring=(String)companyinfo.nameinput.getSelectedItemAt();
        companystore store=new companystore();
        try
        {
            Connection con=store.getConnection();
            Statement st=con.createStatement();
            String sql="delete from company where name='"+namestring+"'";
            st.executeUpdate(sql);
        }
        catch(Exception e){}
        companyinfo.nameinput.removeItem(namestring);
    }
}
```

最后给出朋友删除功能类的程序代码，如下所示：

```
import java.sql.Connection;
import java.sql.Statement;
public class delfriend
{
    delfriend()
    {
        String namestring=(String)friendinfo.nameinput.getSelectedItemAt();
        friendstore store=new friendstore();
        try
        {
            Connection con=store.getConnection();
            Statement st=con.createStatement();
            String sql="delete from comfriend where name='"+namestring+"'";
            st.executeUpdate(sql);
        }
        catch(Exception e){}
        friendinfo.nameinput.removeItem(namestring);
    }
}
```

上述的删除功能主要是通过SQL语句在数据库中直接删除来达到删除数据库中数据的目的

的。接下来将为读者介绍有关更新功能类的设计。

3. 更新功能模块的设计

更新功能要达到两个目的：其一是让被修改后的数据保存到数据库中；其二就是让被修改的数据能够立即保存到界面上。首先来观察有关同学信息的更新模块的设计，其程序代码如下所示：

```
import java.sql.Connection;
import java.sql.Statement;
public class updateclassmate
{
    classmatestore store=new classmatestore();
    Connection con=store.getConnection();
    updateclassmate()
    {
        try
        {
            Statement st=con.createStatement();
            String sex=classinfo.sexyinput.getText();
            String name=(String)classinfo.nameinput.getSelectedItem();
            String address=classinfo.addressinput.getText();
            String homeaddress=classinfo.homeaddressinput.getText();
            String city=classinfo.cityinput.getText();
            String company=classinfo.companyinput.getText();
            String duty=classinfo.dutyinput.getText();
            String salary=classinfo.salaryinput.getText();
            String sql1="update classmate set sex='"+sex+"' where name='"+name+"'";
            // 将需要更新的列以 SQL 语句的形式给出
            String sql2="update classmate set address='"+address+"' where name='"+name+"'";
            String sql3="update classmate set homeaddress='"+homeaddress+"' where name='"+name+"'";
            String sql4="update classmate set city='"+city+"' where name='"+name+"'";
            String sql5="update classmate set company='"+company+"' where name='"+name+"'";
            String sql6="update classmate set duty='"+duty+"' where name='"+name+"'";
            String sql7="update classmate set salary='"+salary+"' where name='"+name+"'";
            st.executeUpdate(sql1);
            // 将上面的 SQL 语句通过 executeUpdate()方法将数据库进行更新
            st.executeUpdate(sql2);
            st.executeUpdate(sql3);
            st.executeUpdate(sql4);
            st.executeUpdate(sql5);
            st.executeUpdate(sql6);
            st.executeUpdate(sql7);
        }
        catch(Exception e){}
    }
}
```

此处的更新功能通过一个 update 的 SQL 语句来直接更新数据库中的内容，同时也会在界面上被保存下来。接下来观察有关同事更新功能模块的设计，其程序代码如下所示：

```
import java.sql.Connection;
import java.sql.Statement;
public class updatecompany
{
    classmatestore store=new classmatestore();
```



```

Connection con=store.getConnection();
updatecompany()
{
    try
    {
        Statement st=con.createStatement();
        String sex=companyinfo.sexinput.getText();
        String name=(String)companyinfo.nameinput.getSelectedItem();
        String address=companyinfo.addressinput.getText();
        String code=companyinfo.codeinput.getText();
        String department=companyinfo.departmentinput.getText();
        String duty=companyinfo.dutyinput.getText();
        String salary=companyinfo.salaryinput.getText();
        String birthday=companyinfo.birthdayinput.getText();
        String sql1="update company set sex='"+sex+"' where name='"+name+"'";
        // 将需要更新的列以 SQL 语句的形式给出
        String sql2="update company set address='"+address+"' where name='"+name+"'";
        String sql3="update company set code='"+code+"' where name='"+name+"'";
        String sql4="update company set department='"+department+"' where name='"+name+"'";
        String sql5="update company set duty='"+duty+"' where name='"+name+"'";
        String sql6="update company set salary='"+salary+"' where name='"+name+"'";
        String sql7="update company set birthday='"+birthday+"' where name='"+name+"'";
        st.executeUpdate(sql1);
        // 将上面的 SQL 语句通过 executeUpdate()方法将数据库进行更新
        st.executeUpdate(sql2);
        st.executeUpdate(sql3);
        st.executeUpdate(sql4);
        st.executeUpdate(sql5);
        st.executeUpdate(sql6);
        st.executeUpdate(sql7);
    }
    catch(Exception e){}
}
}

```

此处的更新功能也是通过一个 update 的 SQL 语句来直接更新数据库中的内容,同时也会在界面上被保存下来。接下来看一下有关朋友更新功能模块的设计,其程序代码如下所示:

```

import java.sql.Connection;
import java.sql.Statement;
public class updatefriend
{
    classmatestore store=new classmatestore();
    Connection con=store.getConnection();
    updatefriend()
    {
        try
        {
            Statement st=con.createStatement();
            String sex=friendinfo.sexyinput.getText();
            String name=(String)friendinfo.nameinput.getSelectedItem();
            String address=friendinfo.addressinput.getText();
            String birthday=friendinfo.birthdayinput.getText();
            String duty=friendinfo.dutyinput.getText();
            String salary=friendinfo.salaryinput.getText();
            String company=friendinfo.companyinput.getText();

```



```

String sql1="update comfriend set sex='"+sex+"' where name='"+name+"'";
String sql2="update comfriend set address='"+address+"' where name='"+name+"'";
String sql3="update comfriend set birthday='"+birthday+"' where name='"+name+"'";
String sql4="update comfriend set company='"+company+"' where name='"+name+"'";
String sql5="update comfriend set duty='"+duty+"' where name='"+name+"'";
String sql6="update comfriend set salary='"+salary+"' where name='"+name+"'";
st.executeUpdate(sql1);
// 将上面的 SQL 语句通过 executeUpdate()方法将数据库进行更新
st.executeUpdate(sql2);
st.executeUpdate(sql3);
st.executeUpdate(sql4);
st.executeUpdate(sql5);
st.executeUpdate(sql6);
}
catch(Exception e){}
}
}

```

此处的更新功能同样也是通过一个 update 的 SQL 语句来实现更新功能的。

4. 查询功能

下面将讲述有关查询功能类的设计，在本软件系统中主要是根据姓名来查询相应的数据信息。下面先看一下同学查询系统，其代码如下：

```

import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class classmatefind
{
    private static final long serialVersionUID = 1L;
    String sql="select * from classmate";
    classmatestore store=new classmatestore();
    Connection con=store.getConnection();
    static JPanel pane;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        pane.add(c,constraints);
    }
    classmatefind()
    {
        pane=new JPanel();
        GridBagLayout lay=new GridBagLayout();
        pane.setLayout(lay);
        JLabel name=new JLabel("姓名");
    }
}

```



```

JLabel sex=new JLabel("性别");
JLabel address=new JLabel("所在地址");
JLabel homeaddress=new JLabel("家庭地址");
JLabel city=new JLabel("所在城市");
JLabel company=new JLabel("所在公司");
JLabel duty=new JLabel("职位");
JLabel salary=new JLabel("薪水");
JLabel namefindlabel=new JLabel("按姓名查询");
JButton namefindbutton=new JButton("查询");
JButton closebutton=new JButton("关闭窗口");
final JTextField sexinput=new JTextField(10);
final JTextField nameinput=new JTextField(10);
final JTextField addressinput=new JTextField(10);
final JTextField homeaddressinput=new JTextField(10);
final JTextField cityinput=new JTextField(10);
final JTextField companyinput=new JTextField(10);
final JTextField dutyinput=new JTextField(10);
final JTextField salaryinput=new JTextField(10);
final JTextField namefind=new JTextField(10);
JLabel title=new JLabel("同学查询结果信息");
JLabel title1=new JLabel("查询系统");
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.weightx=7;
constraints.weighty=8;
add(title,constraints,0,0,4,1);           // 使用网格组布局添加控件
add(name,constraints,0,1,1,1);
add(sex,constraints,0,2,1,1);
add(address,constraints,0,3,1,1);
add(homeaddress,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(sexinput,constraints,1,2,1,1);
add(addressinput,constraints,1,3,1,1);
add(homeaddressinput,constraints,1,4,1,1);
add(city,constraints,2,1,1,1);
add(company,constraints,2,2,1,1);
add(duty,constraints,2,3,1,1);
add(salary,constraints,2,4,1,1);
add(cityinput,constraints,3,1,1,1);
add(companyinput,constraints,3,2,1,1);
add(dutyinput,constraints,3,3,1,1);
add(salaryinput,constraints,3,4,1,1);
add(title1,constraints,0,5,4,1);
add(namefindlabel,constraints,0,6,1,1);
add(namefind,constraints,1,6,1,1);
add(namefindbutton,constraints,2,6,1,1);
add(closebutton,constraints,3,6,1,1);
namefindbutton.addActionListener(new ActionListener()
{ // 通过输入姓名来查询数据库中相应姓名的其他所有信息，并且将之显示在查询窗口的文本框中
    public void actionPerformed(ActionEvent Event)
    {
        String namestring=namefind.getText();
        classmate p=store.getobject(con,namestring);
        String inputsex=p.getsex();
        String inputaddress=p.getaddress();
        String inputhomeaddress=p.gethomeaddress();
    }
}

```



```

String inputcity=p.getcity();
String inputcompany=p.getcompany();
String inputduty=p.getduty();
String inputsalary=p.getsalary();
nameinput.setText(namefind.getText());
sexinput.setText(inputsex);
addressinput.setText(inputaddress);
homeaddressinput.setText(inputhomeaddress);
cityinput.setText(inputcity);
companyinput.setText(inputcompany);
dutyinput.setText(inputduty);
salaryinput.setText(inputsalary);
    }
});
closebutton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        pane.setVisible(false);
    }
});
pane.setVisible(false);
}
}

```

上面程序代码的运行结果如图 20.22 所示。

图 20.22 同学查询系统

接下来看看同事查询系统，其程序代码如下所示：

```

import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class companyfind
{
    private static final long serialVersionUID = 1L;
    String sql="select * from company ";
    companystore store=new companystore();

```



```

Connection con=store.getConnection();
static JPanel pane;
public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
{
    constraints.gridx=x;
    constraints.gridy=y;
    constraints.gridwidth=w;
    constraints.gridheight=h;
    pane.add(c,constraints);
}
companyfind()
{
    pane=new JPanel();
    GridBagLayout lay=new GridBagLayout();
    pane.setLayout(lay);
    JLabel name=new JLabel("姓名");
    JLabel code=new JLabel("工号");
    JLabel department=new JLabel("部门");
    JLabel sex=new JLabel("性别");
    JLabel address=new JLabel("家庭地址");
    JLabel birthday=new JLabel("出生年月");
    JLabel duty=new JLabel("职位");
    JLabel salary=new JLabel("薪水");
    JLabel namefindlabel=new JLabel("按姓名查询");
    JButton namefindbutton=new JButton("查询");
    JButton closebutton=new JButton("关闭窗口");
    final JTextField codeinput=new JTextField(10);
    final JTextField nameinput=new JTextField(10);
    final JTextField departmentinput=new JTextField(10);
    final JTextField sexinput=new JTextField(10);
    final JTextField addressinput=new JTextField(10);
    final JTextField birthdayinput=new JTextField(10);
    final JTextField dutyinput=new JTextField(10);
    final JTextField salaryinput=new JTextField(10);
    final JTextField namefind=new JTextField(10);
    JLabel title=new JLabel("同事查询结果信息");
    JLabel title1=new JLabel("查询系统");
    GridBagConstraints constraints=new GridBagConstraints();
    constraints.fill=GridBagConstraints.NONE;
    constraints.weightx=4;
    constraints.weighty=6;
    add(title,constraints,0,0,4,1);           // 使用网格组布局添加控件
    add(name,constraints,0,1,1,1);
    add(code,constraints,0,2,1,1);
    add(sex,constraints,0,3,1,1);
    add(department,constraints,0,4,1,1);
    add(nameinput,constraints,1,1,1,1);
    add(codeinput,constraints,1,2,1,1);
    add(sexinput,constraints,1,3,1,1);
    add(departmentinput,constraints,1,4,1,1);
    add(birthday,constraints,2,1,1,1);
    add(address,constraints,2,2,1,1);
    add(duty,constraints,2,3,1,1);
    add(salary,constraints,2,4,1,1);
    add(birthdayinput,constraints,3,1,1,1);
    add(addressinput,constraints,3,2,1,1);

```



```
add(dutyinput,constraints,3,3,1,1);
add(salaryinput,constraints,3,4,1,1);
add(title1,constraints,0,5,4,1);
add(namefindlabel,constraints,0,6,1,1);
add(namefind,constraints,1,6,1,1);
add(namefindbutton,constraints,2,6,1,1);
add(closebutton,constraints,3,6,1,1);
namefindbutton.addActionListener(new ActionListener()
{ // 通过输入姓名来查询数据库中相应姓名的其他所有信息, 并且将之显示在查询窗口的文本框中
    public void actionPerformed(ActionEvent Event)
    {
        String namestring=(String)namefind.getText();
        company p=store.getobject(con,namestring);
        String inputcode=p.getcode();
        String inputsex=p.getsex();
        String inputbirthday=p.getbirthday();
        String inputdepartment=p.getdepartment();
        String inputaddress=p.getaddress();
        String inputduty=p.getduty();
        String inputsalary=p.getsalary();
        nameinput.setText(namefind.getText());
        sexinput.setText(inputsex);
        codeinput.setText(inputcode);
        birthdayinput.setText(inputbirthday);
        addressinput.setText(inputaddress);
        departmentinput.setText(inputdepartment);
        dutyinput.setText(inputduty);
        salaryinput.setText(inputsalary);
    }
});
closebutton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        pane.setVisible(false);
    }
});
pane.setVisible(false);
}
```

上面程序代码的运行结果如图 20.23 所示。

同事查询结果信息			
姓名	<input type="text"/>	出生年月	<input type="text"/>
工号	<input type="text"/>	家庭地址	<input type="text"/>
性别	<input type="text"/>	职位	<input type="text"/>
部门	<input type="text"/>	薪水	<input type="text"/>
查询系统			
按姓名查询	<input type="text"/>	<input type="button" value="查询"/>	

图 20.23 同事查询功能类

最后,看看朋友查询功能类的实现,其程序代码如下所示:

```
import java.awt.Component;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class friendfind
{
    private static final long serialVersionUID = 1L;
    friendstore store=new friendstore();
    Connection con=store.getConnection();
    String sql="select * from company ";
    static JPanel pane;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        pane.add(c,constraints);
    }
    friendfind()
    {
        pane=new JPanel();
        GridBagLayout lay=new GridBagLayout();
        pane.setLayout(lay);
        JLabel name=new JLabel("姓名");
        JLabel sex=new JLabel("性别");
        JLabel birthday=new JLabel("出生年月");
        JLabel address=new JLabel("家庭地址");
        JLabel company=new JLabel("所在公司");
        JLabel duty=new JLabel("职位");
        JLabel salary=new JLabel("薪水");
        JLabel namefindlabel=new JLabel("按姓名查询");
        JButton namefindbutton=new JButton("查询");
        JButton closebutton=new JButton("关闭窗口");
        final JTextField nameinput=new JTextField(10);
        final JTextField sexinput=new JTextField(10);
        final JTextField addressinput=new JTextField(10);
        final JTextField birthdayinput=new JTextField(10);
        final JTextField companyinput=new JTextField(10);
        final JTextField dutyinput=new JTextField(10);
        final JTextField salaryinput=new JTextField(10);
        final JTextField namefind=new JTextField(10);
        JLabel title=new JLabel("朋友查询结果信息");
        JLabel title1=new JLabel("查询系统");
        GridBagConstraints constraints=new GridBagConstraints();
        constraints.fill=GridBagConstraints.NONE;
        constraints.weightx=4;
        constraints.weighty=6;
```



```

add(title,constraints,0,0,4,1);           // 使用网格组布局添加控件
add(name,constraints,0,1,1,1);
add(sex,constraints,0,2,1,1);
add(address,constraints,0,3,1,1);
add(birthday,constraints,0,4,1,1);
add(nameinput,constraints,1,1,1,1);
add(sexinput,constraints,1,2,1,1);
add(addressinput,constraints,1,3,1,1);
add(birthdayinput,constraints,1,4,1,1);
add(company,constraints,2,1,1,1);
add(duty,constraints,2,2,1,1);
add(salary,constraints,2,3,1,1);
add(companyinput,constraints,2,4,1,1);
add(birthdayinput,constraints,3,1,1,1);
add(dutyinput,constraints,3,2,1,1);
add(salaryinput,constraints,3,3,1,1);
add(title1,constraints,0,5,4,1);
add(namefindlabel,constraints,0,6,1,1);
add(namefind,constraints,1,6,1,1);
add(namefindbutton,constraints,2,6,1,1);
add(closebutton,constraints,3,6,1,1);
namefindbutton.addActionListener(new ActionListener()
{ // 通过输入姓名来查询数据库中相应姓名的其他所有信息，并且将之显示在查询窗口的文本框中
    public void actionPerformed(ActionEvent Event)
    {
        String namestring=(String)namefind.getText();
        friend p=store.getobject(con,namestring);
        String inputsex=p.getsex();
        String inputbirthday=p.getbirthday();
        String inputaddress=p.getaddress();
        String inputcompany=p.getcompany();
        String inputduty=p.getduty();
        String inputsalary=p.getsalary();
        nameinput.setText(namestring);
        sexinput.setText(inputsex);
        birthdayinput.setText(inputbirthday);
        addressinput.setText(inputaddress);
        companyinput.setText(inputcompany);
        dutyinput.setText(inputduty);
        salaryinput.setText(inputsalary);
    }
});
closebutton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        pane.setVisible(false);
    }
});
pane.setVisible(false);
}
}

```

上面程序代码的运行结果如图 20.24 所示。

图 20.24 朋友查询功能类

20.3.4 其他模块的设计

在本软件系统的帮助系统中主要有两个类：一个是版本信息类，另一个是帮助文档类。而帮助文档类使用的是树组件。本节主要的目的是让读者复习如何使用树组件，并且如何处理其事件。

1. 版本信息的设计

下面先来看一下版本信息类的代码设计，如下所示：

```
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.Toolkit;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class version extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static final int WIDTH=200;
    static final int HEIGHT=100;
    JFrame versionframe;
    public void add(Component c,GridBagConstraints constraints,int x,int y,int w,int h)
    {
        constraints.gridx=x;
        constraints.gridy=y;
        constraints.gridwidth=w;
        constraints.gridheight=h;
        add(c,constraints);
    }
    version()
    {
        versionframe=new JFrame("通讯录系统版本信息");
        versionframe.setContentPane(this);
        versionframe.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        versionframe.setSize(WIDTH,HEIGHT);
        Toolkit kit=Toolkit.getDefaultToolkit();
```



```

Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
versionframe.setLocation(x,y);
versionframe.setVisible(true);
GridBagConstraints constraints=new GridBagConstraints();
constraints.fill=GridBagConstraints.NONE;
constraints.anchor=GridBagConstraints.EAST;
constraints.weightx=3;
constraints.weighty=2;
JLabel label1=new JLabel("通讯录信息系统版本信息");
JLabel label2=new JLabel("此软件版本 version 1.0");
JLabel label3=new JLabel("作者: 王鹏");
add(label1,constraints,1,0,2,1);
add(label2,constraints,1,1,2,1);
add(label3,constraints,2,2,1,1);
}
}

```

上面程序代码的运行结果如图 20.25 所示。

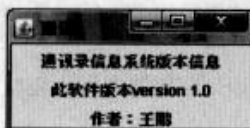


图 20.25 版本信息的设计界面

2. 帮助文档系统的设计

接下来将为读者展示帮助文档类的代码，如下所示：

```

import java.awt.Dimension;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTextArea;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
public class help
{
    static final int WIDTH=700;
    static final int HEIGHT=400;
    JTree tree;
    DefaultMutableTreeNode root;
    DefaultMutableTreeNode node1;
    DefaultMutableTreeNode node2;
    DefaultMutableTreeNode node3;
    DefaultMutableTreeNode node4;
    static JTextArea text;
}

```



```

help()
{
    JFrame frame=new JFrame();
    frame.setTitle("通讯录系统帮助文档");
    frame.setSize(WIDTH,HEIGHT);
    root=new DefaultMutableTreeNode("通讯录系统帮助文档");
    node1=new DefaultMutableTreeNode("如何操作同学通讯系统");
    node2=new DefaultMutableTreeNode("如何操作同事通讯系统");
    node3=new DefaultMutableTreeNode("如何操作朋友通讯系统");
    node4=new DefaultMutableTreeNode("如何操作查询系统");
    root.add(node1);
    root.add(node2);
    root.add(node3);
    root.add(node4);
    DefaultMutableTreeNode leafnode=new DefaultMutableTreeNode("如何使用同学信息模块");
    node1.add(leafnode);
    leafnode=new DefaultMutableTreeNode("如何使用同学通讯模块");
    node1.add(leafnode);
    leafnode=new DefaultMutableTreeNode("如何使用同事信息模块");
    node2.add(leafnode);
    leafnode=new DefaultMutableTreeNode("如何使用同事通讯模块");
    node2.add(leafnode);
    leafnode=new DefaultMutableTreeNode("如何使用朋友信息模块");
    node3.add(leafnode);
    leafnode=new DefaultMutableTreeNode("如何使用朋友通讯模块");
    node3.add(leafnode);
    tree=new JTree(root);
    JScrollPane scrollpane=new JScrollPane(tree);
    JPanel p1=new JPanel();
    JPanel p2=new JPanel();
    JSplitPane splitPane = new JSplitPane ();
    splitPane.setOneTouchExpandable (true);
    splitPane.setContinuousLayout (true);
    splitPane.setPreferredSize (new Dimension (100,200));
    splitPane.setOrientation (JSplitPane.HORIZONTAL_SPLIT);
    splitPane.setLeftComponent (p1);
    splitPane.setRightComponent (p2);
    splitPane.setDividerSize (3);
    splitPane.setDividerLocation(200);
    frame.setContentPane(splitPane);
    p1.add(scrollpane);
    frame.setVisible(true);
    tree.addMouseListener(new MouseHandle());
    text=new JTextArea();
    p2.add(text);
}
}
class MouseHandle extends MouseAdapter
{
    public void mousePressed(MouseEvent e)
    {
        String nodeName;
        try{// 当单击鼠标时,在右边会显示出相应的数据信息
            JTree tree = (JTree)e.getSource();
            int rowLocation = tree.getRowForLocation(e.getX(), e.getY());
            TreePath treepath = tree.getPathForRow(rowLocation);

```



```

TreeNode treenode = (TreeNode) treepath.getLastPathComponent();
nodeName = treenode.toString();
help.text.setText("要想查询如何使用"+nodeName+"模块相关知识, 请查阅光盘");
// 在右边文本区中所显示的数据信息
}
catch(NullPointerException ne){}
}
}

```

上面程序代码的运行结果如图 20.26 所示。

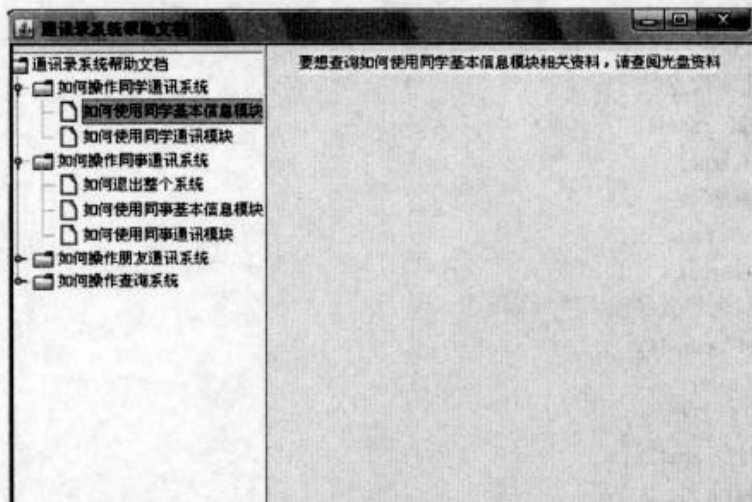


图 20.26 帮助文档界面的设计

上面的帮助文档系统主要是采用了树组件来创建的, 单击任何一项帮助时, 在右边都会显示相应的文字说明。当然这个帮助文档系统只是为了让读者能够熟悉这种结构的创建方法, 所以右边的说明文字稍微简单一点。

20.3.5 TabbedPane 容器框架的设计

此软件会将基本信息放到一个 TabbedPane 容器中去, 所以本节将讲述如何设计这个容器, 因为这个框架不涉及到对象数据细节, 所以无论是朋友通讯系统、同事通讯系统还是同学通讯系统, 操作方法都是一样的。于是, 可以通过设计成一个类来被其他多个程序调用。其设计的程序代码如下所示:

```

import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
public class tabpane extends JPanel
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    JPanel panel1;
    static JPanel panel2;
    static JPanel panel3;
    static JPanel panel4;
    static JPanel panel5;
    static JTabbedPane tp;

```



```
public tabpane()
{
    setLayout(new BorderLayout());
    tp=new JTabbedPane();
    panel1 = new JPanel ();
    panel2 = new JPanel ();
    panel3 = new JPanel ();
    panel4 = new JPanel ();
    panel5 = new JPanel ();
    panel1.setLayout(new BorderLayout());
    tp.addTab("panel1", panel1);
    tp.setEnabledAt(0,true);
    tp.setTitleAt(0,"基本信息");
    tp.addTab ("panel2", panel2);
    tp.setEnabledAt (1, true);
    tp.setTitleAt (1,"照片");
    tp.addTab ("panel3", panel3);
    tp.setEnabledAt (2, true);
    tp.setTitleAt (2,"兴趣与爱好");
    tp.addTab ("panel4", panel4);
    tp.setEnabledAt(0,true);
    tp.setTitleAt(3,"日常习惯");
    tp.addTab ("panel5", panel5);
    tp.setEnabledAt(4,true);
    tp.setTitleAt(4,"评价");
    tp.setPreferredSize (new Dimension (500,200));
    tp.setTabPlacement (JTabbedPane.TOP);
    tp.setTabLayoutPolicy (JTabbedPane.SCROLL_TAB_LAYOUT);
    add("Center",tp);
    tp.setVisible(false);
}
```

上面程序代码的运行结果如图 20.27 所示。

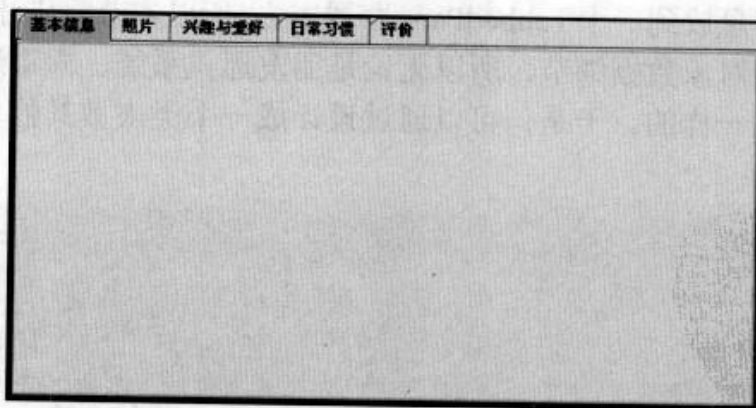


图 20.27 TabbedPane 容器类的实现

20.3.6 主菜单的设计

本小节将为读者介绍如何进行主菜单的设计，这个主菜单的设计要点有两点：第一就是菜单项的创建；第二就是菜单项的事件处理。由于前面的篇幅已讲述实现软件的所有功能，并且将它们设计成了类，所以现在只需要在程序中调用它们即可，其程序代码如下所示：

```
import javax.swing.*;
```



```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class mainframe
{
    private static final long serialVersionUID = 1L;
    static final int WIDTH=600;
    static final int HEIGHT=400;
    JPopupMenu pop;
    JMenuItem item2;
    JFrame f;
    JMenuItem item1;
    JPanel p;
    JToolBar bar;
    JPanel p1;
    JPanel p2;
    JPanel p3;
    tabpane tab1;
    tabpane tab2;
    tabpane tab3;
    JButton closebutton1;
    JButton closebutton2;
    JButton closebutton3;
    JButton addbutton1;
    JButton addbutton2;
    JButton addbutton3;
    JButton delbutton1;
    JButton delbutton2;
    JButton delbutton3;
    JButton updatebutton1;
    JButton updatebutton2;
    JButton updatebutton3;
    public mainframe()
    {
        f=new JFrame("我的通讯录系统主界面");
        JMenuBar menubar1=new JMenuBar();
        p=new JPanel();
        f.setContentPane(p);
        f.setJMenuBar(menubar1);
        // 创建菜单系统
        JMenu menu1=new JMenu("同学通讯系统");
        JMenu menu2=new JMenu("同事通讯系统");
        JMenu menu3=new JMenu("朋友通讯系统");
        JMenu menu4=new JMenu("查询系统");
        JMenu menu5=new JMenu("帮助系统");
        menubar1.add(menu1);
        menubar1.add(menu2);
        menubar1.add(menu3);
        menubar1.add(menu4);
        menubar1.add(menu5);
        item1=new JMenuItem("同学基本信息系统");
        item2=new JMenuItem("同学联系方式系统");
        JMenuItem item3=new JMenuItem("退出通讯录系统");
        JMenuItem item4=new JMenuItem("同事基本信息系统");
        JMenuItem item5=new JMenuItem("同事联系方式系统");
        JMenuItem item6=new JMenuItem("朋友基本信息系统");
```



```
JMenuItem item7=new JMenuItem("朋友联系方式系统");
JMenuItem item8=new JMenuItem("同学查询系统");
JMenuItem item9=new JMenuItem("同事查询系统");
JMenuItem item10=new JMenuItem("朋友查询系统");
JMenuItem item11=new JMenuItem("版本信息");
JMenuItem item12=new JMenuItem("帮助信息");
menu1.add(item1);
menu1.addSeparator();
menu1.add(item2);
menu1.addSeparator();
menu1.add(item3);
menu2.add(item4);
menu2.addSeparator();
menu2.add(item5);
menu3.add(item6);
menu3.addSeparator();
menu3.add(item7);
menu4.add(item8);
menu4.addSeparator();
menu4.add(item9);
menu4.addSeparator();
menu4.add(item10);
menu5.add(item11);
menu5.addSeparator();
menu5.add(item12);
JButton button1 = new JButton("同学查询系统");
JButton button2 = new JButton("同事查询系统");
JButton button3 = new JButton("朋友查询系统");
closebutton1=new JButton("关闭");
closebutton2=new JButton("关闭");
closebutton3=new JButton("关闭");
addbutton1=new JButton("添加同学信息数据");
addbutton2=new JButton("添加同事信息数据");
addbutton3=new JButton("添加朋友信息数据");
delbutton1=new JButton("删除同学信息数据");
delbutton2=new JButton("删除同事信息数据");
delbutton3=new JButton("删除朋友信息数据");
updatebutton1=new JButton("更新同学信息数据");
updatebutton2=new JButton("更新同事信息数据");
updatebutton3=new JButton("更新朋友信息数据");
// 将功能按钮组件添加到容器中
p1=new JPanel();
p2=new JPanel();
p3=new JPanel();
p1.setLayout(new FlowLayout());
p1.add(closebutton1);
p1.add(addbutton1);
p1.add(delbutton1);
p1.add(updatebutton1);
p2.add(closebutton2);
p2.add(addbutton2);
p2.add(delbutton2);
p2.add(updatebutton2);
p3.add(closebutton3);
p3.add(addbutton3);
p3.add(delbutton3);
```



```

p3.add(updatebutton3);
bar = new JToolBar();
bar.add(button1);
bar.add(button2);
bar.add(button3);
BorderLayout bord = new BorderLayout();
p.setLayout(bord);
p.add("North",bar);
f.setVisible(true);
f.setSize(WIDTH,HEIGHT);
Toolkit kit=Toolkit.getDefaultToolkit();
Dimension screenSize=kit.getScreenSize();
int width=screenSize.width;
int height=screenSize.height;
int x=(width-WIDTH)/2;
int y=(height-HEIGHT)/2;
f.setLocation(x,y);
item1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 将 JTabbedPane 组件添加到框架容器内的中间位置
        // 将包含添加、删除和更新组件的容器添加到框架容器的最底部
        tab1=new tabpane();
        p.add("Center",tab1);
        tab1.panel1.add(new classinfo());
        p.add("South",p1);
        tabpane.tp.setVisible(true);
        p1.setVisible(true);
    }
});
item2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 当单击查询系统中的同学查询系统，会弹出前面设计的同学联系方式类
        classmatecommunication cc=new classmatecommunication();
        p.add("Center",cc.pane);
        cc.pane.setVisible(true);
    }
});
item3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 设计一个对话框系统，如果需要退出系统的话，则主框架消失
        int i=JOptionPane.showConfirmDialog(null,"是否真的需要退出系统",
            "退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            f.dispose();
        }
    }
});
item4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 将 JTabbedPane 组件添加到框架容器内的中间位置
        //将包含添加、删除和更新组件的容器添加到框架容器的最底部
    }
});

```



```
        tab2=new tabpane();
        p.add("Center",tab2);
        tab2.panel1.add(new companyinfo());
        p.add("South",p2);
        tabpane.tp.setVisible(true);
        p2.setVisible(true);
    }
});
item5.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 当单击查询系统中的同事查询系统, 会弹出前面设计的同事联系方式类
      companycommunication cc1=new companycommunication();
      p.add("Center",cc1.pane);
      cc1.pane.setVisible(true);
    }
});
item6.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 将 JTabbedPane 组件添加到框架容器内的中间位置
      // 将包含添加、删除和更新组件的容器添加到框架容器的最底部
      tab3=new tabpane();
      p.add("Center",tab3);
      tab3.panel1.add(new friendinfo());
      p.add("South",p3);
      tabpane.tp.setVisible(true);
      p3.setVisible(true);
    }
});
item7.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 当单击查询系统中的朋友查询系统, 会弹出前面设计的朋友联系方式类
      friendcommunication fc=new friendcommunication();
      p.add("Center",fc.pane);
      fc.pane.setVisible(true);
    }
});
item8.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 当单击查询系统中的同学查询系统, 会弹出前面设计的同学查询类
      classmatefind cf=new classmatefind();
      p.add("Center",cf.pane);
      cf.pane.setVisible(true);
    }
});
item9.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 当单击查询系统中的同事查询系统, 会弹出前面设计的同事查询类
      companyfind cf1=new companyfind();
      p.add("Center",cf1.pane);
      cf1.pane.setVisible(true);
    }
});
```



```
});
item10.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 当单击查询系统中的朋友查询系统, 会弹出前面设计的朋友查询类
        friendfind ff=new friendfind();
        p.add("Center",ff.pane);
        ff.pane.setVisible(true);
    }
});
item11.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 通过版本信息的构造器来创建一个版本信息的框架
        new version();
    }
});
item12.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        new help();
    }
});
button1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        classmatefind cf=new classmatefind();
        p.add("Center",cf.pane);
        cf.pane.setVisible(true);
    }
});
button2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        companyfind cfl=new companyfind();
        p.add("Center",cfl.pane);
        cfl.pane.setVisible(true);
    }
});
button3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        friendfind ff=new friendfind();
        p.add("Center",ff.pane);
        ff.pane.setVisible(true);
    }
});
closebutton1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {

```



```
        tab1.setVisible(false);
        p1.setVisible(false);
    }
});
closebutton2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        tab2.setVisible(false);
        p2.setVisible(false);
    }
});
closebutton3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        tab3.setVisible(false);
        p3.setVisible(false);
    }
});
addbutton1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器来创建一个添加功能的框架窗口
        new addclassmate();
    }
});
addbutton2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器来创建一个添加功能的框架窗口
        new addcompany();
    }
});
addbutton3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器来创建一个添加功能的框架窗口
        new addfriend();
    }
});
delbutton1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器创建一个删除功能模块
        int i=JOptionPane.showConfirmDialog(null,"是否真的删除此项",
        退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            new delclassmate();
        }
    }
});
delbutton2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
```



```
{ // 通过构造器创建一个删除功能模块
    int i=JOptionPane.showConfirmDialog(null,"是否真的删除此项","
        退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
    if(i==0)
    {
        new delcompany();
    }
}

});
delbutton3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器创建一个删除功能模块
        int i=JOptionPane.showConfirmDialog(null,"是否真的删除此项","
            退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            new delfriend();
        }
    }
});
updatebutton1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器创建更新功能的模块
        int i=JOptionPane.showConfirmDialog(null,"是否真的要更新此项","
            退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            new updateclassmate();
        }
    }
});
updatebutton2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    { // 通过构造器创建更新功能的模块
        int i=JOptionPane.showConfirmDialog(null,"是否真的要更新此项","
            退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            new updatecompany();
        }
    }
});
updatebutton3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent Event)
    {
        // 通过构造器创建更新功能的模块
        int i=JOptionPane.showConfirmDialog(null,"是否真的要更新此项","
            退出确认对话框",JOptionPane.YES_NO_CANCEL_OPTION);
        if(i==0)
        {
            new updatefriend();
        }
    }
});
```


上面程序代码的运行结果如图 20.28 所示。

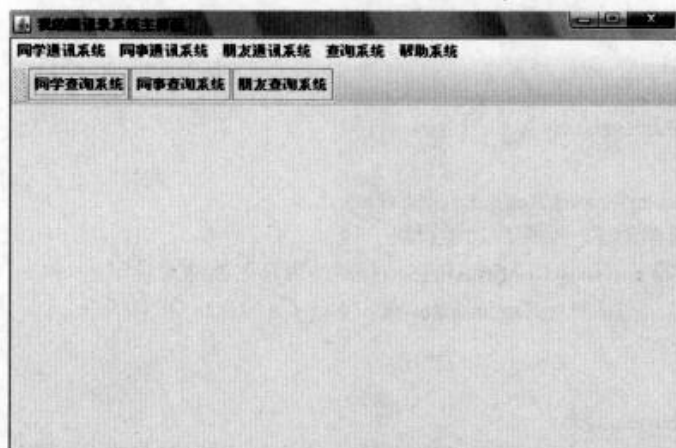


图 20.28 主菜单程序

当打开每个菜单后，将弹出不同的菜单项，如图 20.29~20.33 所示。

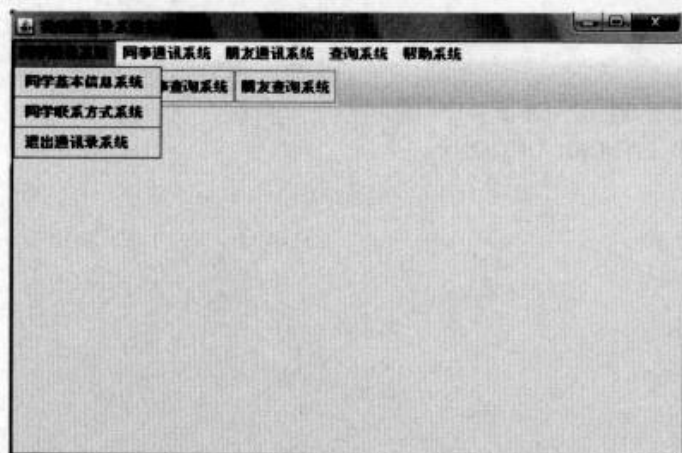


图 20.29 同学通讯系统

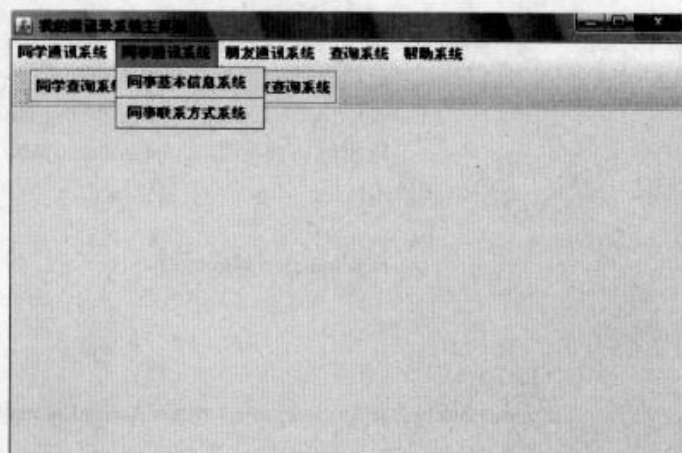


图 20.30 同事通讯系统

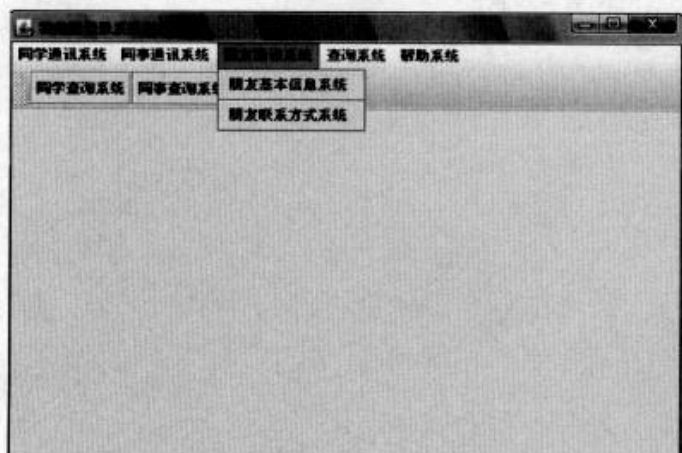


图 20.31 朋友通讯系统

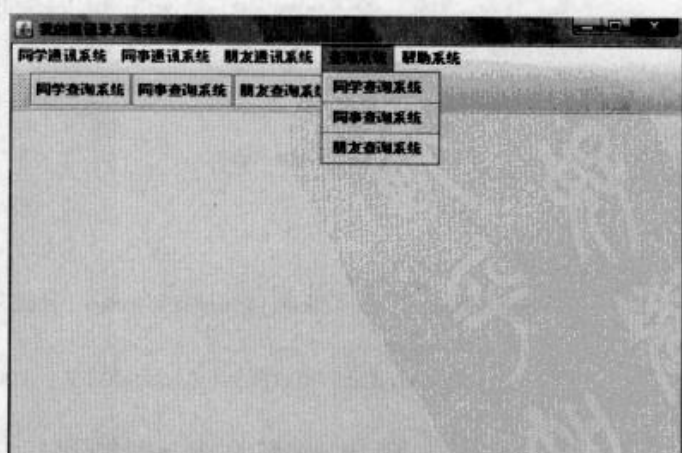


图 20.32 查询系统

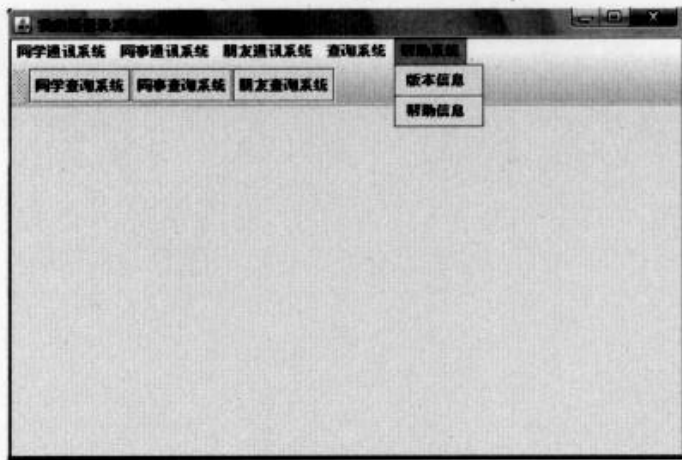


图 20.33 帮助系统

另外，在主菜单中还添加了三个功能快捷键，其用途与查询菜单中的每个查询项的功能相同。

20.4 本章小结

本章主要为读者介绍了如何开发一个 Swing 图形程序：首先在整个系统的登录界面输入正确的用户名和密码后，就可以进入到主菜单窗口。在主菜单上可以实现不同的功能，其中有基本信息、联系方式信息、查询信息、帮助信息等。在基本信息中又包括几种功能，如添加、删除、更新等。而整个系统的数据信息依靠的是底层数据库系统，其底层数据库系统将所有数据以基本类的形式存储到 Vector 数据结构中去，系统通过对 Vector 数据结构中的数据进行操作来达到对底层数据库的操作。

限于篇幅，整个软件设计的比较粗糙，希望读者能够在此基础上进一步完善，以将其设计成专业的、合理的、实用性强的软件。

Java Swing

图形界面开发与案例详解

本书内容

- 内容全面、浅显易懂、逐步深入、逐步掌握
- 几乎涉及到Java Swing图形开发所需要掌握的所有常用知识点
- 结合105个实例和77个习题使读者能够对常用的组件知识进行巩固和熟练
- 通过1个综合实例，使读者进一步巩固所学知识，提高综合应用的能力

读者对象

- Java开发图形界面设计的初学者
- 具有一定Java基础的编程人员
- 参加二级Java等级考试的考试人员



下载资源：

可通过访问<http://www.booksaga.com>下载本书所有实例的源代码

使用者指引

入门

进阶

专家

上架指导：计算机/Java Swing界面开发
封面设计：张仁伟

ISBN 978-7-302-18904-6



9 787302 189046 >

定价：49.00元